Update: 2009-07-03 23:13:54 (Fri)
Edit

MCARaTS User's Guide (Version 0.10)

This is a guide for users of MCARaTS (Monte Carlo Atmospheric Radiative Transfer Simulator).

(C) Copyright 2009 MCARaTS project. All rights reserved.

Edit

Contents

1. Introduction
2. Installation
3. Model descriptions
4a. Input to mcarats executable (a): Structure of the namelist input file...
4b. Input to mcarats executable (b): Namelist mcarWld_nml_init...
4c. Input to mcarats executable (c): Namelist mcarWld_nml_job...
4d. Input to mcarats executable (d): External input file Sca_inpfile...
5. Radiative transfer simulation
6. Output from mcarats executable
7. Utilities
8. Using MCARaTS
9. References

A single page that includes all documents is available here.

Edit

Next / Return

- 1. Introduction
  - What is MCARaTS
  - Features of the RTM
  - Output quantities from the RTM
  - 3D radiative transfer
  - MCARaTS softwares

Edit

1. Introduction

What is MCARaTS

MCARaTS is a general purpose radiative transfer model (RTM). It simulates the three-dimensional radiative transfer (3DRT) in atmosphere-ocean-land system using the Monte Carlo methods. The model can be applied to the ultraviolet to microwave spectral range. However, polarization is not taken into account. The codes can be used for simulations of radiative energy budget and quasi-observation with optical instruments.
A major purpose to use the software could be to simulate accurate radiative quantities as virtual observation data obtained with optical instrument (or human eyes) for cloud-containing atmosphere over land/ocean surface. The other objective is to investigate the three-dimensional radiative transfer within inhomogeneous media. Volume rendering is a new feature. Approximate RTE solvers are incorporated mainly for visualization of 3-D volumetric data. Thus MCARaTS can be used as a visualization tool.
Input to the RTM should be optical properties of the atmosphere and surfaces (i.e. extinction coefficients, single scattering albedos, phase functions, and sruface BRDFs).

Features of the RTM

Can be summarized as follows:
What's good : Easy to use, fast, and parallelized
Basic algorithm : Forward-propagating Monte Carlo photon transport algorithm
Radiative transfer solvers

- Fully-3-D radiative transfer (F3D)
- Partially-3-D radiative transfer (P3D)
- Independent column approximation (ICA)

Input

- Three-dimensionally inhomogeneous atmosphere
- Inhomogeneous surface modeled by BRDF models
  1. Lambertian model
  2. Diffuse-specular-mixture (DSM) model
  3. Rahman-Pinty-Verstraete (RPV) model
  4. Li-Sparse-Ross-Thick (LSRT) model

Output quantities

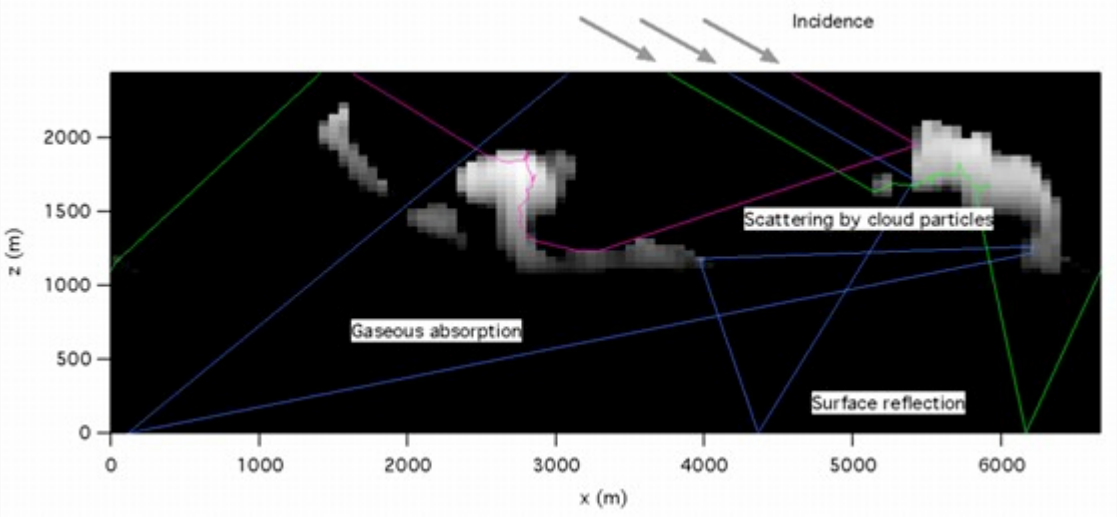
- Radiative fluxes and heating rates
- Radiances with pathlength statistics
- Pseudo-radiances for quick visualization of 3-D volumetric data

Numerical techniques

- The maximum cross section technique applied to super-cells
- The local estimation method
- Variance reduction techniques
  - Collision-forcing method for optically-thin media
  - Flexible truncation of forward peak of phase function
  - Numerical diffusion (a denoising technique)
  - The Russian roulette method
- Parallelization using MPI

Requirements

- Linux/UNIX-like operating system
- Fortran 90/95 compiler
- MPI is optionally needed for parallelization.



Output quantities from the RTM

Various types of radiative quantities can be calculated.

- Upward/downward/direct fluxes at layer interfaces (averaged over finite areas)
- Volume-averaged heating rates (averaged over finite volumes)
- Radiances possibly with pathlength statistics
  1. Radiances averaged over solid angles, at some locations
  2. Radiances averaged over horizontal areas, for some specific directions
  3. Radiances averaged over horizontal areas and solid angles

For 1 and 2, a visualization mode can be chosen.

- Pathlength statistics (optional)
  1. Layer average pathlength normalized by vertical layer depth: Layer air mass factors
  2. Pathlengths averaged with weighting by user-specified functions: Air mass factors
  3. Histogram of pathlength integrated from the source emission to the radiance detector: Time-resolved radiance distributions

The above can be calculated pixel-by-pixel for each user-specified radiance.
Local fluxes could not be output in the latest version. Instead, radiances averaged over solid angle can be calculated. If the FOV is a hemisphere, the averaged radiance can be easily converted to irradiance on any inclined plane or hemispherical actinic flux density.

3D radiative transfer

In previous studies on transfer of energy, several kinds of 3D radiative transfer (3DRT) solvers have been developed, such as finite difference and characteristics methods. We can see a variety of 3DRT codes in the I3RC project for example. The Monte Carlo method is a kind of simple and powerful solvers. Its accuracy is well known due to its physically correct basis and a number of tests. That is why the Monte Carlo method has been frequently used as reference to validate other methods.
For a study on 3DRT, an exact method and two approximations to solve 3D radiative transfer are incorporated in the MCARaTS codes:

- Fully-3-D radiative transfer (F3D)
- Partially-3-D radiative transfer (P3D)
- Independent column approximation (ICA)

ICA neglects horizontal transfer of radiation and applies 1D radiative transfer (1DRT) to atmospheric columns independently. Using the ICA mode, we can compare 3D and 1D schemes strikingly even for very complicated model atmospheres, not being affected by different property descriptions (e.g., constant or trilinear extinction in each cell), area/volume-averaging or method of phase function truncation etc.

MCARaTS softwares

The package includes softwares for the followings:

- Radiative transfer solvers,
- Processing utility tools for the I/O data.

The processing tools can process the data file output from the RT solvers. See also the corresponding manual pages for details.

Edit

Next / Return

Next / Return

- 2. Installation
  - Compatibility
  - Unpacking
  - Configuration
    - Power users: Encapsulated real types
    - Parallelization
  - How to install
  - Tests
    - Big endian?
  - Porting to Mac: For Mac users

Edit

2. Installation

Compatibility

MCARaTS v0.10 works in UNIX or Linux-like environment. Source codes conform to the Fortran 90/95 standards. So they can be compiled with virtually any compiler that comforms to the standards.
Tested sets are as follows:

- Gfortran, Mac OS X with Intel Core 2 Duo
- G95 (Fortran compiler), Cygwin, Windows
- Intel Fortran Compiler version 9, Redhat Linux WS3 with Intel Xeon
- Intel Fortran compiler version 10.1, SuSE Linux, SGI Altix4700

Older versions also worked under the following conditions, but not tested for MCARaTS v0.10:

- GCC (GNU Compiler Collection) version 2.95 or higher
- Fujitsu Fortran compiler version 3.0
- UNIX on the HP (Compaq) AlphaServerSC with HP (Compaq) Fortran compiler
- Mac OS X on an Apple PPC Macintosh with GCC version 3.2 or higher
- SUN Solaris on a UltraSPARC workstation with SUN Fortran 77 version 4.2.1.
- UNIX on SGI Origin2000 with SGI Fortran compiler.

Your reports about porting to the other type of computer system are welcome. Reports of computational speed are also welcome.

Unpacking

The softwares are archived to a tar-gzipped file. To unpack it,

```
% gunzip mcarats-*.tar.gz
% tar xf mcarats-*.tar
```

then you will find a directory, mcarats-*, that includes the source codes and sample files as follows:

```
ReadMe    : Read this first
src/      : Source codes
shl/      : Shell scripts.
examples/ : Example I/O files
```

Configuration

Several system-dependent variables for compiling with "make" tool should be configured before installation. Edit first a file:

```
./src/Common.mk
```

Compiler command and options and sevral system-dependent variables can be set in the above file.

Power users: Encapsulated real types

For power users, there are some advanced options for configuration before compiling.
MCARaTS codes uses real types encapsulated in two global constants, R_ and RD_, defined in src/hparx/globals.f90, for ALL floating point variables and values. The precision of the types can be easily configured by editing two source files,

```
./src/hparx/globals.f90
./src/mcarats/globals_MPI.F90
```

It is very easy to modify real kinds, R_ and RD_. This affects ALL floating point variables and values used in ALL codes. Note that high precision real kinds requires more memory and could require more CPU time, depending on CPU architecture (e.g., for 32-bits architechture).

Parallelization

If you are building MPI-parallelized codes, you might need to edit two files

```
./src/mcarats/Makefile
./src/mcarats/inc_mpi.f90
```

for system-specific configuration of MPI libraries.

How to install

To build the executable files, execute the following commands:

```
% cd src
% make
% make install
% cd ..
```

After installation you will find executable files in ./bin directory. If you have any problem, please report details of your problem to the developer team. Reports on problems and corresponding resolutions are especially appreciated.
To build the parallelization codes, for example,

```
% cd src
% make clean
% make USE_MPI=1
% make install USE_MPI=1 BINDIR=../bin-mpi
% cd ..
```

Radiative transfer codes are parallelized with MPI, but other codes are not. See corresponding Makefile, src/mcarats/Makefile, for details.

Tests

It is highly recommended for users to test the codes for the sample cases. For that purpose, a shell script is provided.

```
$ cd ./examples
$ csh job_test
```

The results will be found with filenames, test_*. They can be compared with reference files, out_*. Using larger number of photons for simulation, you may get more accurate result that might be closer to the reference files. The number of photons are defined in the shell script. Note that the result will differ second by second, because the random number generator for Monte Carlo simulation is initialized with a time the user starts the simulation. Usages and functions of the codes are easily learned from the tests.
Also, recommended is a statistics comparison:

```
$ cd ./examples
$ ../bin/mcarats 1e5 0 conf_rs0067 test_rs0067
$ ../bin/bin_stat test_rs0067.ctl
$ ../bin/bin_stat  out_rs0067.ctl
```

Reference results in out_rs0067:

SZA(deg.)	Average radiance
0	4.8122E-01
60	3.6292E-01

Big endian?

Before running the above shell script for tests, you might have to check the default kind of endian. There are two types of endian, little and big endians. The binary data in examples in the mcarats package is encoded in little endian, which is compatible with Intel architechture. If your computer system assumes big endian, you might convert the binary data first. Fortunately, there are many useful tools to convert endian. They would be easily found on the web.
There is an easier method to test mcarats, which provides an option to convert the endian of input binary data within the code on the fly. See here (in Chapter 4) for that option.

Porting to Mac: For Mac users

Mac OS X provides a good platform for numerical computing with gcc and some "Xcode" tools (make and ar etc.). But any Fortran compiler is not included by default. To compile Fortran source code, try to install one of free Fortran compilers, GFortran and G95, or any comercial compiler if you like.
Install Xcode first. After setting up your shell environment, you are ready to compile any Fortran codes. For better numerical works, possibly X11 (or some alternative X server software) will be convenient. And GrADS and ImageMagick would be useful. Try also Fink.

Next / Return

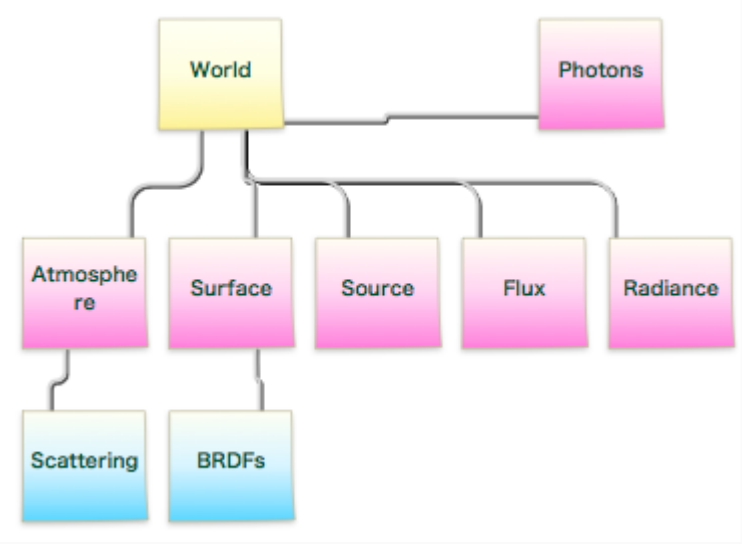
- 3. Model descriptions
  - General
  - Geometry
  - Surface
  - Atmosphere
  - Phase functions
  - Integrators
  - Algorithms
    - Methods and techniques
  - General concepts assumed in the MCARaTS RT solver
    - Experiment
    - Computational jobs
    - Radiative sources
    - Data sets in input property data files
    - Scattering order

Edit

3. Model descriptions

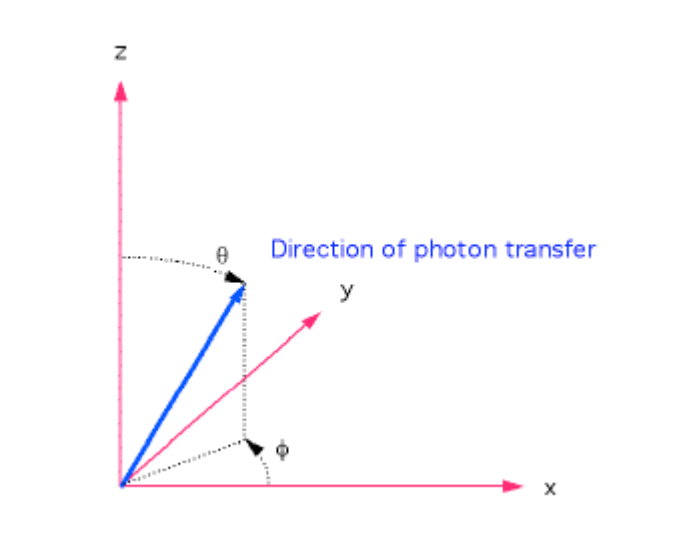
General

MCARaTS numerical world is composed of several modules. Some concepts behind them are described below.



Geometry

The position of photon is defined in the 3D Cartesian coordinate system. The direction of photon transport is defined by zenith (polar) angle (theta) and azimuth angle (phi) from the X-axis, as in the following figure.



The incident and emergent directions for pixel radiances can be specified by the user. The zenith and azimuth angles of source light-transport direction are defined as theta0 and phi0, respectively. Similarly, emergent zenith and azimuth angles are defined as theta1 and phi1, respectively. The horizontal boundary condition is cyclic. The vertical boundaries z=zmin and zmax correspond to the bottom and top, respectively, of the atmosphere (BOA and TOA, respectively).

Surface

The surface is modeled as one of the following models

- black (no reflection)
- Lambertian model
- Diffuse-specular mixture model: diffuse reflection is Lambertian, and specular reflection is Fresnellian for rough surface facets.
- Rahman-Pinty-Verstraete BRDF model
- Li-Sparse-Ross-Thick BRDF model

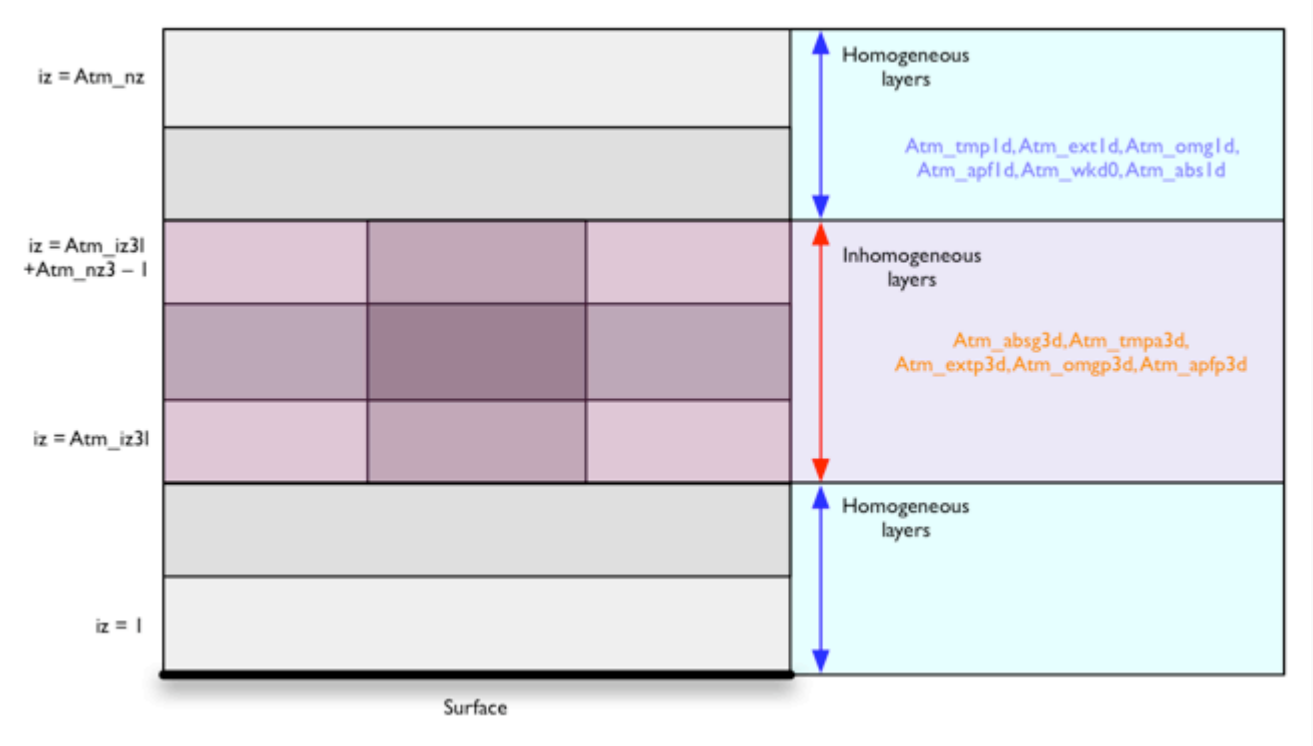
Surface model and its parameters can vary two-dimensionally.

Atmosphere

The model atmosphere is divided into cell volumes three-dimensionally. The number of layer is defined as nz.
The user can specify cloud layers ("3D layers") arbitrarily with horizontal inhomogeneity. Other layers are assumed to be horizontally homogeneous. The 3D layers are divided into cells in the vertical and horizontal directions. The numbers of 3D cells in x-, y-, and z-directions are defined as nx, ny, and nz3, respectively.
Modeled atmospheric media include the followings:

- Scattering constituents: For each one constituent, extinction coefficient, single scattering albedo and a specification parameter for phase function are arbitrarily specified. The user defines multiple kinds of phase function, which can be Rayleigh scattering phase function, Henyey-Greenstein function, or a function interpolated from user-specified database.
- Gaseous absorbing constituents: Modeled by correlated k-distribution.

All properties can be specified in 1D (for layers) and/or in 3D (for 3D cells). The atmosphere and surface system is schematically illustrated in the figure below.



Phase functions

One method for phase function data representation is to prepare the phase functions for

- multiple kinds of aerosol polydispersions,
- multiple types of hydrometeors (e.g., cloud water and ice, drizzle droplets, rain water, snow, and graupel)
  - different shapes,
  - different size distributions, and/or
  - different effective radii and effective variances.

Total number of phase functions could be 100 to 10,000, which is significantly smaller than the number (1,000,000 or larger) of atmospheric cells in usual cases. To obtain highly accurate radiance estimates, every phase functions should be tabulated for 200 or more points of scattering angles.
In the mcarats, the phase functions should be given in a database file, and 3D distribution of a phase function specification parameter should be given. The latter data should be specified cell-by-cell, so that used computer memory should be a word length multipled by the number of atmospheric cells.
Thus, the strategy to prepare the phase function database and 3D distribution of phase function specification parameter is economical for limited computer memory!
In addition, the phase function specification parameter is not digitalized in the latest version. The phase functions used in the code are interpolated from the database table!

Integrators

Various radiative quantities can be computed by the code:

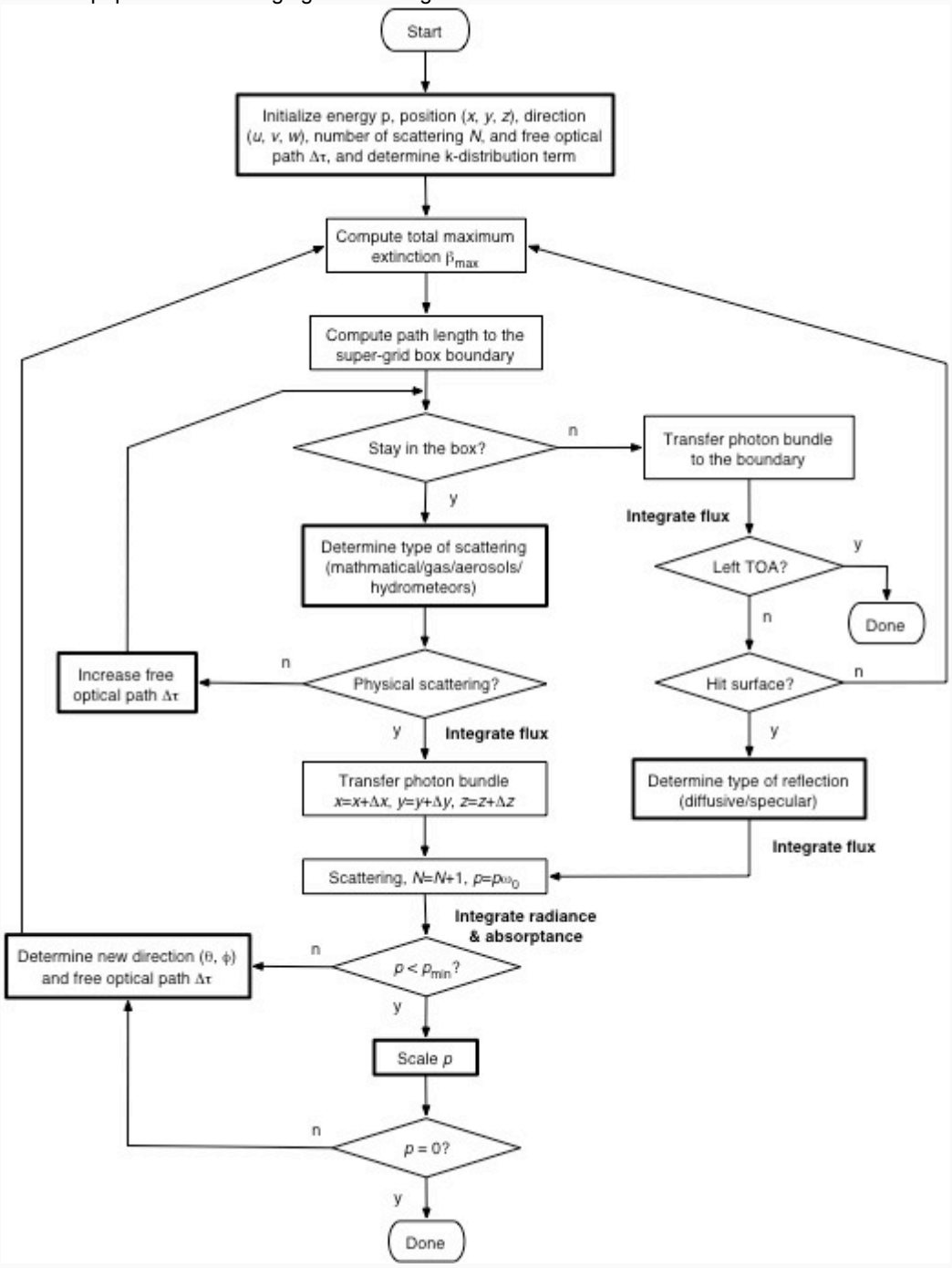
- Upward/downward/direct fluxes at layer interfaces, averaged over finite areas
- Voxel-averaged 3-D heating rates
- Radiances
  - Radiances averaged over finite horizontal areas at arbitrary layer interfaces
  - Local radiances averaged over finite solid angles, looking from arbitrary points
  - Radiances averaged over finite horizontal areas and for planes at arbitrary layer interfaces
- Pathlength statistics for the radiances
  - Layer air mass factors
  - Air mass factors for user-specified weighting functions
  - Time-resolved radiance histogram

For radiances, the user should specify the location of view point, field of view (FOV), and FOV-center direction with zenith and azimuth angles, a method of angular pixel mapping, and numbers of angular pixels.

Algorithms

Used algorithms are described in Iwabuchi (2006, J. Atmos. Sci.). The model uses Monte Carlo methods for simulating photon trajectories and samples fluxes, heating rates, radiances, and other radiometric quantities. The local estimation method is used for

radiance averaged over some specific area or over specific solid angle. The maximum cross section method is used for acceleration of photon tracing in inhomogeneous media. Other methods useful for variance reduction are also used and documented in the scientific paper. The following figure is an algorithm flow chart.



Methods and techniques

Various methods and techniques are used in the code, for better performance. Some of them are described in Iwabuchi & Kobayashi (2008, FRCGC Tech. Rep.).

- The maximum cross section technique applied to super-cells
- The local estimation method
- Collision-forcing method for optically-thin media
- Truncation approximations for forward-peaked phase functions
- Numerical diffusion
- The Russian roulette method
- Parallelization using MPI

The method of truncation of forward peak of phase function is incorporated. The truncation fraction adaptively increases with the order of scattering (Iwabuchi & Suzuki 2009 JQSRT).

General concepts assumed in the MCARaTS RT solver

Experiment

One experiment is performed by one namelist input file and one output data file. More precisely, six I/O files can be used in a single

experiment:

Intent	File
in	namelist file
in	scattering database file
in	atmospheric property data file
in	surface property file
out	result output file
out	GrADs control file for the output

Several files are optional. Details are described in Chapter 4.

Computational jobs

In a single experiment, multiple jobs can be performed. These jobs are defined in the namelist input file. For each job, atmospheric and surface data, radiative sources, and target quantities can be configured by the user. Scattering phase function database is initialized first, and used commonly for all jobs.

Radiative sources

In a single job, multiple radiative sources can be set, and results will be obtained for them. A type of radiative source can be local source with arbitrary angular spread, solar source incident from the top of atmosphere, thermal source, or solar plus thermal source. The source type and property of source can be different for each source configuration. For example, three radiative sources can be set:

1. Solar source with zenith angle of 60 degrees
2. Solar source with zenith angle of 65 degrees
3. Thermal source at a wavelength 11 micrometer

Data sets in input property data files

In the input data files for atmospheric and surface properties, multiple data sets can be stored. The files can be binary or text format. In mcarats simulations, data sets to be read in can be easily chosen by data set indexes in the namelist file, if the file is in binary format. This data set can be different for respective computational jobs.
IMPORTANT NOTE : Binary format is highly recommended, because the data can be read in faster than text format. In the future versions, it is likely that only binary format files could be accepted.

Scattering order

Is 0 for direct beam. After the first-order scattering, the order is set to 1. In the current version, some numerical techniques are controled depending on the scattering order.

Next / Return

- 4. Input to "mcarats" code
  - Structure of the namelist input file
  - Full list of input variables
  - Notes on namelist input

Edit

4. Input to "mcarats" code

The executable for radiative transfer simulation is named as "mcarats". A single run of mcarats executable is called here an experiment. For one experiment, four input files can be used:

Variable	Description
-	namelist file
Sca_inpfile	scattering database file
Atm_inpfile	atmospheric property data file
Sfc_inpfile	surface property file

The namelist input file controls general configurations of the experiment. Other data files are optional, and used for large array data such as scattering phase functions and surface and 3-D atmospheric properties.

Edit

Structure of the namelist input file

The namelist file should contain one initialization part and job specification parts for multiple jobs:

```
Initialization part
Job specification part for job 1
Job specification part for job 2
Job specification part for job 3
...
```

The following subsections describe details of each input parameters for the namelists and the external files. There are many input parameters. In addition, this document could not be complete. To learn how to set up the parameters for the user's purpose, the easiest way might be to explore the example cases, for which input data files are included in the package.
Cases are provided for 3-D inhomogeneous, overcast stratocumulus cloud over homogeneous surface

- Remote sensing ("rs") experiment: calculation of pixel-averaged, reflected radiances at the top of atmosphere (TOA)
- Camera image ("ci") experiment: calculation of local angle-averaged radiances looking the 3-D scene
- Heating rate ("hr") experiment: calculation of 3-D distributions of pixel-averaged fluxes and volume-averaged heating rates

Edit

Full list of input variables

```
namelist /mcarWld_nml_init/Wld_mverb, Wld_jseed, Wld_mbswap, Wld_mtarget, Wld_moptim, Wld_njob
  namelist /mcarSca_nml_init/Sca_inpfile, Sca_mfmt, Sca_npf, Sca_nangi, Sca_nanci, &
       & Sca_ipsta, Sca_ipwid, Sca_nchi, Sca_qtfmax, Sca_ntg
  namelist /mcarAtm_nml_init/Atm_inpfile, Atm_mfmt, Atm_iz3l, Atm_nx, Atm_ny, Atm_nz, &
       & Atm_nz3, Atm_nkd, Atm_npld, Atm_np3d, Atm_mtprof, Atm_nqlay
  namelist /mcarSfc_nml_init/Sfc_inpfile, Sfc_mfmt, Sfc_mbrdf, Sfc_nxb, Sfc_nyb, Sfc_nsco, Sfc_nsuz
  namelist /mcarSrc_nml_init/Src_nsrc
  namelist /mcarFlx_nml_init/Flx_mflx, Flx_mhrt, Flx_diff0, Flx_diff1
  namelist /mcarRad_nml_init/Rad_mrkind, Rad_mpmap, Rad_mplen, Rad_nrad, Rad_nxr, &
       & Rad_nyr, Rad_nwf, Rad_ntp, Rad_tpmin, Rad_tpmax
  namelist /mcarVis_nml_init/Vis_mrend, Vis_epserr, Vis_fpsmth, Vis_nqhem, Vis_nqlay, Vis_fatten
  namelist /mcarPho_nml_init/Pho_iso_SS, Pho_iso_tru, Pho_iso_max, Pho_wsml, Pho_wmin, Pho_wmax, &
       & Pho_wfac, Pho_pfpeak
  namelist /mcarWld_nml_job/Wld_rmlcf, Wld_nplcf, Wld_dtaumin
  namelist /mcarAtm_nml_job/Atm_idread, Atm_dx, Atm_dy, Atm_zgrd0, Atm_tmpld, Atm_extld, Atm_omgld, &
       & Atm_apfld, Atm_wkd0, Atm_absld, Atm_fextld, Atm_fext3d, Atm_fabsld, Atm_fabs3d, &
       & Atm_taumin, Atm_fsupg, Atm_fsupi, Atm_nzzsup
  namelist /mcarSfc_nml_job/Sfc_idread, Sfc_tmp, Sfc_mtype, Sfc_param, Sfc_nqpot, Sfc_rrmax, &
       & Sfc_rrexp, Sfc_nudsm, Sfc_nurpv, Sfc_nulsrt
  namelist /mcarSrc_nml_job/Src_wlen, Src_dwlen, Src_flx, Src_qmax, Src_the, Src_phi, Src_xpos, &
       & Src_ypos, Src_zloc, Src_apsize, Src_mtype, Src_mphi
  namelist /mcarRad_nml_job/Rad_mrproj, Rad_difr0, Rad_difrl, Rad_zetamin, Rad_frmod, Rad_xpos, &
       & Rad_ypos, Rad_wfunc0, Rad_zloc, Rad_rmin0, Rad_rmax0, Rad_the, Rad_phi, Rad_psi, Rad_umax, &
       & Rad_vmax, Rad_qmax, Rad_apsize
```

In the followings, variable names, types and shapes of variables, default values, and explanations are described.

Edit

Notes on namelist input

The namelist I/O is a feature of Fortran 90. The namelists for mcarats input include many parameters, but the the user does not need to specify all variables' values. This makes the Fortran's namelist flexible and easy to use. For example, when the user gives no value for every variables, then default initialization data will be used. Thus, content of the input file could be as the followings:

```
&mcarWld_nml_init /
&mcarSca_nml_init /
&mcarAtm_nml_init /
&mcarSfc_nml_init /
&mcarSrc_nml_init /
&mcarFlx_nml_init /
&mcarRad_nml_init /
&mcarVis_nml_init /
&mcarPho_nml_init /
&mcarWld_nml_job /
&mcarAtm_nml_job /
&mcarSfc_nml_job /
&mcarSrc_nml_job /
&mcarRad_nml_job /
```

Default initialization data are defined in source codes in ./src/mcarats/mcar*.?90, and they are customizable before compiling. The user can partially set values for one variable in the namelist. For example, a sentence,

```
Atm_abs1d(1:10, 1)=4.0e-5,  3.0e-5,  2.0e-5,  1.0e-5,  5*1.0e-6
```

sets values for a part of the 2-D array variable Atm_abs1d(iz,ikd), which may be declared as a very large array in the source code. Another feature of Fortran namelist is a freedom of ordering of each member variable in a namelist block. Any order is acceptable in one namelist block. For details, see Fortran language reference for the compiler.

Edit

Next / Return

User's Guide v0.10 > 4b

Next / Return

- Namelist mcarWld_nml_init
  - Wld_mverb
  - Wld_jseed
  - Wld_mbswap
  - Wld_mtarget
  - Wld_moptim
  - Wld_njob
- Namelist mcarSca_nml_init
  - Sca_inpfile
  - Sca_mfmt
  - Sca_npf
  - Sca_nskip
  - Sca_nanci
  - Sca_nangi
  - Sca_ntg -technical parameter-
  - Sca_nchi -technical parameter-
  - Sca_qtfmax -technical parameter-
  - Note: Approximation or not?
- Namelist mcarAtm_nml_init
  - Atm_inpfile
  - Atm_mfmt
  - Atm_np1d
  - Atm_np3d
  - Atm_nx, Atm_ny, Atm_nz
  - Atm_iz3l
  - Atm_nz3
  - Atm_nkd
  - Atm_mtprof
  - Atm_nqlay -technical parameter-
- Namelist mcarSfc_nml_init
  - Sfc_inpfile
  - Sfc_mfmt
  - Sfc_nxb, Sfc_nyb
  - Sfc_mbrdf
  - Sfc_nsco -technical parameter-
  - Sfc_nsuz -technical parameter-
- Namelist mcarSrc_nml_init
  - Src_nsrc
- Namelist mcarFlx_nml_init
  - Flx_mflx
  - Flx_mhrt
  - Flx_diff0, Flx_diff1 -technical parameter-
- Namelist mcarRad_nml_init
  - Rad_mrkind
  - Rad_mpmap
  - Rad_mplen

- Rad_nrad
- Rad_nxr, Rad_nyr
- Rad_nwf
- Rad_ntp
- Rad_tpmin, Rad_tpmax
- Namelist mcarVis_nml_init
  - Vis_mrend
  - Vis_fatten
  - Vis_epserr -technical parameter-
  - Vis_fpsmth -technical parameter-
  - Vis_nqhem -technical parameter-
  - Vis_nqlay -technical parameter-
- Namelist mcarPho_nml_init
  - Pho_iso_SS
  - Pho_iso_max -technical parameter-
  - Pho_iso_tru -technical parameter-
  - Pho_wsml -technical parameter-
  - Pho_wmin, Pho_wmax -technical parameter-
  - Pho_wfac -technical parameter-
  - Pho_pfpeak -technical parameter-

Edit

Namelist mcarWld_nml_init

Wld_mverb

```
integer,  save :: Wld_mverb = 0
```

Flag for a verbose mode. Controls verboseness of the executable, (de)activating printing messages. This would be useful for testing and debugging. Options are as follows:

```
= 0 : quiet (no message will be printed)
  1 : yes
  2 : more
  3 : most (for debug)
```

Wld_jseed

```
integer,  save :: Wld_jseed = 0
```

Seed for random number generator (0 for automatic). Should be positive integer, >= 1 and <= 2**31 - 1 if the user specify the seed. This would be useful in some cases (e.g., when debugging).
If Wld_jseed == 0, then the seed is automatically set by the code, using the start date and time. In MPI-parallelized mode, processor element (PE) rank index is also used for the automatic seed, so that every PEs could have completely uncorrelated results. By automatic seed, different results will be obtained trial by trial.

Wld_mbswap

```
  integer,  save :: Wld_mbswap = 0
```

Flag for byte swapping for binary input files (0=no, 1=yes). Input data files can be in binary or text format. For binary files, the code mcarats has capability of instant byte swapping just after reading the data stream. Binary data representation differs by computer architecture. If Wld_mbswap == 1, then every 4 bytes for a datum are rearranged in the reverse order. This is useful when the code mcarats is run on a machine with a different architecture from that of machine on which the input files were created.

Wld_mtarget

```
  integer,  save :: Wld_mtarget = 1
```

Flag for target quantities (computing mode). Options are as follows:

```
  = 1 : flux mode, computing fluxes and heating rates by MC methods
    2 : radiance mode, computing radiances by MC methods
    3 : volume-rendering mode, computing quasi-radiances (or some signals) by volume renc
```

The code mcarats computes only one type of target radiative quantities: This is different from previous versions.
If different types of target quantities (e.g., flux and radiance) are needed for some reasons, then they should be calculated in independent two experiments, by running the mcarats code two times with different I/O files.

Wld_moptim

```
  integer,  save :: Wld_moptim = 2
```

Flag for optimization of calculation techniques. This selects one of built-in presets for technical parameters, which controls trade-off between used computer resources and accuracy (quality) of obtained results. Many technical parameters might not be easy for users to tune. Presets were well tuned by the developer.
Possible options for Wld_moptim are as follows:

```
 = -2 : no tuning (use default optimization)
   -1 : no optimization (deactivate all default optimizations)
    0 : unbiased optimization (deactivate all biasing optimizations)
    1 : conservative optimizations (possibly for smaller biases)
    2 : standard optimizations (recommended)
    3 : quick-and-dirty optimizations (for speed when small biases are acceptable)
```

The quality setting can differ by whether the user's goal is accuracy or computation speed. Wld_moptim=0 might be useful for debugging and/or tuning experiments.
If Wld_moptim is not -1, then all values given in the input file are overwritten by the built-in preset. The user can select a preset by Wld_moptim, and then modify the optimization by specifying each parameter value, that overrides the preset, in subsequent namelists.
The following parameters are set by Wld_moptim, automatically:

```
Wld_dtaumin, Wld_rmlcf, Wld_nplcf, Wld_emtmin,
Sca_nchi,
Atm_taumin, Atm_fsupg, Atm_fsupi, Atm_nzzsup
```

```
Sfc_nudsm, Sfc_nurpv, Sfc_nulsrt, Sfc_rrmax, Sfc_rrexp,
Flx_diff0, Flx_diff1, Flx_diff0, Flx_diff1,
Rad_zetamin, Rad_difr0, Rad_difr1, Rad_difr0, Rad_difr1,
Pho_wmin, Pho_wmax, Pho_wfac
```

The preset values are given in source codes, src/mcarats/mcar*.?90.

Wld_njob

```
integer,  save :: Wld_njob = 1
```

Number of jobs in a single experiment. Here, an experiment corresponds to one namelist input file and one output file. The code mcarats can process multiple jobs in an experiment (a new feature). For each job, a set of namelists mcar*_job should be given consequently.

Edit

Namelist mcarSca_nml_init

Sca_inpfile

```
character(256), save :: Sca_inpfile = ' '
```

Filename for external file input of tabulated phase function data. Leave it as default ' ' if no external input file is needed.
The file name should have relative path from a directory where the namelist input file exists. For example, when Sca_inpfile = 'mcarats_phs', and the namelist input file is './exp1/mcarats_conf', then actual name of the phase function input file will be './exp1/mcarats_phs'.
See also description of the data in Sca_inpfile.

Sca_mfmt

```
integer,  save :: Sca_mfmt = 1
```

Flag for format (0=text, 1=binary) of the phase function input file.
If 0, the file should be in sequential access, text format.
If 1, the file should be in direct access, binary format.
IMPORTANT NOTE : Binary format is highly recommended, because the data can be read in faster than text format. In the future versions, it is likely that only binary format files could be accepted.

Sca_npf

```
integer,  save :: Sca_npf = 1
```

Number of tabulated phase functions that are saved in the data file.

Sca_nskip

```
integer,  save :: Sca_nskip = 0
```

Number of first data record lines to be skipped. Used only when the data file is in binary format. If this is 100, for example, then first 100 record lines are not read in, line 101 should be angle data, and subsequent (Sca_npf) lines will be read in as phase function data.

Sca_nanci

```
integer,  save :: Sca_nanci = 0
```

If the file is in binary format, this is the number of ancillary data in a record in the data file. Note that the ancillary data is not used in the mcarats code. Number of data recorded in one record line should be

```
Sca_nangi + Sca_nanci
```

If the file is in text format, then Sca_nanci is the number of blocks of ancillary data in one data set. Details are described in subsequent section.

Sca_nangi

```
integer,  save :: Sca_nangi = 1000
```

If Sca_mfmt == 0 (text format), Sca_nangi is the maximum number of angles for the input table. The number of angles could differ each other phase functions recorded in the file.
If Sca_mfmt == 1 (binary format), Sca_nangi is the number of angles for the input table. The number of angles should be the same for all phase functions recorded in the file.

Sca_ntg -technical parameter-

```
integer,  save :: Sca_ntg  = 20000
```

Number of interpolated table grid points for angles and probabilities. In the mcarats code, every phase functions are interpolated for fine angular grid. For random scattering angle determination, cumulative probability functions are computed, and inverse reference tables are created; Scattering angles are tabulated w.r.t. discrete probabilities. Sca_ntg is the number of their grid points.

Sca_nchi -technical parameter-

```
integer,  save :: Sca_nchi = 4
```

Number of orders for truncation approximation (>= 2) for anisotropic phase function. The approximation order, ichi, follows

```
ichi=1 : no truncation approximation
ichi=[2,nchi] : truncation approximation
```

For higher orders, forward peaks of phase functions are geometrically truncated with larger truncation fractions. The approximation order is determined adaptively depending on photon scattering history. Larger value for Sca_nchi could result in

smaller bias, larger size of used memory, and larger noise in results.
Default value is well tuned by Wld_moptim and generally optimal.

Sca_qtfmax -technical parameter-

```
real(R_), save :: Sca_qtfmax = 20.0_R_
```

Max scattering angle (degrees) for geometrical truncation of phase functions. Order of truncation approximation is adaptively determined depending on history of photon trajectory. For highest orders, the max angle specified here is used.
Default value is well tuned by Wld_moptim and generally optimal.

Note: Approximation or not?

Some parameters described above activate the truncation approximations which may improve numerical efficiency (trade-off between MC noise and computation time). However the approximations may introduce bias error, which could not be negligible in specific user's problem. If the user would not like to use the approximation (that can be biasing) for some reason, then a required configuration is as Sca_qtfmax=0.

Edit

Namelist mcarAtm_nml_init

Atm_inpfile

```
character(256), save :: Atm_inpfile = ' ' ! file name for input of 3D optical propertie
```

Filename of an external file of 3-D atmospheric model data. Leave it as default ' ' if no 3D atmospheric data are available. 1D atmospheric data can still be given by the namelist mcarAtm_job. The file name should have relative file name path from the directory where the namelist file exists. The file should differ from Sca_inpfile and Sfc_inpfile.
See also description of the data in Atm_inpfile.

Atm_mfmt

```
integer,  save :: Atm_mfmt = 1
```

Flag for format (0=text, 1=binary) of the atmospheric input file.
If 0, the file should be in sequential access, text format.
If 1, the file should be in direct access, binary format.
IMPORTANT NOTE : Binary format is highly recommended, because the data can be read in faster than text format. In the future versions, it is likely that only binary format files could be accepted.

Atm_np1d

```
integer,  save :: Atm_np1d = 1
```

Number of scattering components specified with 1-D profile. 1D atmospheric data can still be given by the namelist mcarAtm_nml_job. A single component can be a mixture of gases, aerosols and hydrometeors or a single bin of size distribution.

Atm_np3d

```
integer,  save :: Atm_np3d = 1
```

Number of scattering components specified with 3-D distribution. The 3-D data are given for horizontally inhomogeneous layers, from the data file (Atm_inpfile). A single component can be a mixture of gases, aerosols and hydrometeors or a single bin of size distribution.

Atm_nx, Atm_ny, Atm_nz

```
integer,  save :: Atm_nx   = 1   ! # of X grid points
integer,  save :: Atm_ny   = 1   ! # of Y grid points
integer,  save :: Atm_nz   = 1   ! # of Z grid points (layers)
```

Numbers of grids along x, y, and z-axis, respectively, in the atmosphere.

Atm_iz3l

```
integer,  save :: Atm_iz3l = 1
```

Lowest layer index of horizontally inhomogeneous layers. Should be >= 1 and <= Atm_nz. Layers for the z-index iz=Atm_iz3l to Atm_iz3l + Atm_nz3 - 1 are defined as horizontally inhomogeneous, and for these layers 3-D data of optical properties are given. See also Atm_nz3.
See here (in Chapter 3) for how homogeneous and inhomogeneous layers are defined.

Atm_nz3

```
integer,  save :: Atm_nz3  = 0
```

Number of horizontally inhomogeneous layers, which is defined for the z-grid index (layer index)

```
iz = Atm_iz3l to Atm_iz3l + Atm_nz3 - 1.
```

See also Atm_iz3l.

Atm_nkd

```
integer,  save :: Atm_nkd  = 1
```

Number of terms for the k-distribution used as the gaseous absorption model. Give Atm_nkd=1 for monochromatic computation.

Atm_mtprof

```
integer,  save :: Atm_mtprof = 0
```

Flag for temperature profile input. Defines a method to specify temperature data, as

follows:

```
0: temperature data are given for each layer (or at layer mid-levels)
1: temperature data are given for each layer interface
```

Required number of z-axis levels for temperature data is

```
Atm_nz for Atm_mtprof=0, or
Atm_nz+1 for Atm_mtprof=1.
```

Temperature data are used only when the source includes thermal emission (Src_mtype = 2 or 3).
If Atm_mtprof = 0, then the temperature at layer interfaces will be calculated by cubic interpolation within the code. In both cases, Planck function profile within each layer is assumed to be log-linear (i.e. log(B) is linear to the altitude, where B is Planck function).
See also Atm_tmpa1d and Atm_tmpa3d.

Atm_nqlay -technical parameter-

```
integer,  save :: Atm_nqlay = 5
```

Number of Gaussian quadrature points per layer, used in the local estimation.
The optical thickness along the line of sight can be calculated by Gaussian quadrature (i.e., approximately). This is activated if the ray could penetrate too many cells in a single layer. This happens for a very slant ray or for atmospheric cells with very large vertical-to-horizontal aspect ratio. Exact ray tracing is time consuming under such conditions. Instead, optical thickness along path segment within a layer is calculated by a Gaussian quadrature with Atm_nqlay points. By limiting the number of quadrature points, Gaussian quadrature is efficient even if the ray is very oblique.
Default value is well tuned and generally optimal.

Edit

Namelist mcarSfc_nml_init

Sfc_inpfile

```
character(256), save :: Sfc_inpfile = ' '
```

Filename of external input file for surface model data. Leave it as default ' ' if no external input file is needed. The name should have relative path from a directory where the configuration file exists.
Surface properties can be specified by some variables in namelist mcarSfc_job or by the external input file. Input data from the external file will override on input the namelist mcarSfc_job.
The file should differ from Sca_inpfile and Atm_inpfile.
See also description of the data in Sfc_inpfile.

Sfc_mfmt

```
integer,  save :: Sfc_mfmt = 1
```

Flag for format (0=text, 1=binary) of the surface input file.
If 0, the file should be in sequential access, text format.

If 1, the file should be in direct access, binary format.
IMPORTANT NOTE : Binary format is highly recommended, because the data can be read in faster than text format. In the future versions, it is likely that only binary format files could be accepted.

Sfc_nxb, Sfc_nyb

```
integer,  save :: Sfc_nxb = 1  ! # of X grid points
integer,  save :: Sfc_nyb = 1  ! # of Y grid points
```

Numbers of X and Y-pixels for the surface properties. This is now independent of the atmosphere. For example, if Sfc_nxb = 1 and Sfc_nyb = 1, then surface properties are constant in the full domain.

Sfc_mbrdf

```
integer,  save :: Sfc_mbrdf(4) = (/1, 1, 1, 1/)
```

Flags of on/off status for BRDF models. The four elements correspond to Lambertian, DSM, RPV, and LSRT BRDF models, respectively. Set as 0 if one BRDF model is not used in the simulation. By doing so, computer memory is economically used. If the memory use is not matter, then set all as 1.

Sfc_nsco -technical parameter-

```
integer,  save :: Sfc_nsco = 60
```

Number of cos(Q0) grid points for surface coefficient table. For each surface finite element, this number of coefficients are stored in a table. The coefficients definitions differ by a type of BRDF.

Sfc_nsuz -technical parameter-

```
integer,  save :: Sfc_nsuz = 200
```

Number of cos(Q0) grid points for albedo LUTs. Used for look-up tables for a few variables, such as black-sky albedo and incident-direction-dependent maximum BRDF value.

Edit

Namelist mcarSrc_nml_init

Src_nsrc

```
integer,  save :: Src_nsrc = 1
```

Number of radiative sources. Simulation results will be obtained for multiple radiative sources in each single job. Here, for one source, a type (solar, thermal, or local source), location, direction, and source power can be specified.

Edit

Namelist mcarFlx_nml_init

Flx_mflx

```
integer,  save :: Flx_mflx = 1
```

Flag for calculating flux density calculations.

```
0: no flux output
1: at top and bottom of the atmosphere (TOA and BOA, respectively)
2: at interfaces of all layers except internal interfaces of 3-D inhomogeneous layers
3: at interfaces of all layers
```

The number of interfaces at which fluxes are computed will be

```
0    for Flx_mflx=0,
2    for Flx_mflx=1,
Atm_nz - Atm_nz3 + 2    for Flx_mflx=2,
Atm_nz + 1    for Flx_mflx=3.
```

See also Atm_nz and Atm_nz3.

Flx_mhrt

```
integer,  save :: Flx_mhrt = 1
```

Flag for calculating heating rate calculations (0=off, 1=on). If Flx_mhrt=1, then heating rates are computed for all Atm_nz layers.

Flx_diff0, Flx_diff1 -technical parameter-

```
real(R_), save :: Flx_diff0 = 1.0_R_
real(R_), save :: Flx_diff1 = 0.1_R_
```

Artificial diffusion coefficients for flux and heating rate sampling.
Default values are generally optimal.

Edit

Namelist mcarRad_nml_init

Rad_mrkind

```
integer,   save :: Rad_mrkind = 2
```

Kind of radiance calculated.

```
= 0 : no radiance calculation
  1 : 1st kind, local radiance averaged over solid angle
  2 : 2nd kind, radiance averaged over pixel (horizontal cross section of atmospheric cc
  3 : 3rd kind, radiance averaged over solid angle and horizontal plane
```

In the volume rendering mode (Wld_mtarget = 3), Rad_mrkind should be 1 or 2.
Specifications of each radiance are given in namelist mcarRad_job.

For the 1st kind, location in the 3D and FOV of the detector and pixel mapping method should be specified.
For the 2nd kind, rectangular pixels are defined on a horizontal plane, and radiance in a specific direction is calculated.
For the 3rd kind, radiances are averaged over a horizontal plane and also averaged over respective solid angles. A pixel mapping method can be chosen in the same way as the 1st kind. This kind of radiance is calculated by sampling power of photons that penetrates the target area and registering it in appropriate angular bin.
Note that a "pixel" of radiance output here corresponds to a finite solid angle (1st and 3rd kind) or a finite area on a plane (2nd kind).

Rad_mpmap

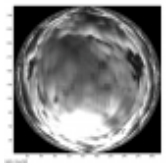
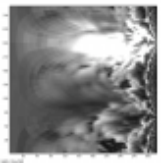
```
integer,   save :: Rad_mpmap = 1
```

A method for pixel mapping, used for 1st or 3rd kind of radiance. Pixel coordinates are here defined by U and V. Definitions of them differs by a kind of radiance.

```
= 1 : rectangular (U = theta * cos(phi), V = theta * sin(phi))
  2 : polar (U = theta, V = phi)
```

where theta is polar angle, and phi is azimuth angle, in the camera coordinate, which are defined by rotation parameters specified in the namelist mcarRad_nml_job.
Photo-like image can be obtained by Rad_mpmap=1.
Here is an example:

Rad_mpmap=1 2

Rad_mplen

```
integer,   save :: Rad_mplen = 0
```

Method of calculation of pathlength statistics.

```
= 0 : (npr = 0)  no calculation of pathlength statistics
  1 : (npr = Atm_nz) layer-by-layer pathlength distribution (layer air mass factor)
  2 : (npr = Rad_nwf) average pathlengths with weighting functions (column air mass fact
  3 : (npr = Rad_ntp) histogram of total, integrated pathlength (time-resolved histogram
```

As above, one of three kinds of pathlength information can be calculated. Number of output elements, npr, differs by the kind. See also Rad_nwf and Rad_ntp. Pathlength statistics can be calculated, not depending on Rad_mrkind, the kind of radiance.

Rad_nrad

```
integer,  save :: Rad_nrad = 0
```

Number of radiance arrays calculated. One radiance array here corresponds to one

location and one view region (or direction). If 0, no radiance is calculated.

Rad_nxr, Rad_nyr

```
integer,  save :: Rad_nxr = 1
integer,  save :: Rad_nyr = 1
```

Number of pixels along U and V coordinates, respectively, for radiance images. See Rad_mpmap for the radiance U and V pixel coordinates' definition.

Rad_nwf

```
integer,  save :: Rad_nwf = 1
```

Number of weighting functions used for column air mass factor. The weighting functions are specified in namelist mcarRad_job and used only when Rad_mplen=2.

Rad_ntp

```
integer,  save :: Rad_ntp = 100
```

Number of total-pathlength bins. Used only when Rad_mplen=3. Total pathlength is calculated by integrating photon flight length between the source emission point and the detector sampling point. , and radiance contribution is accumulated in appropriate pathlength bin each time of radiance sampling.
See also Rad_tpmin and Rad_tpmax.

Rad_tpmin, Rad_tpmax

```
real(R_), save :: Rad_tpmin = 0.0_R_
real(R_), save :: Rad_tpmax = 1.0e+5_R_
```

Minimum and maximum limits, respectively, of total pathlength. Used only when Rad_mplen=3. The pathlength range is divided into bins, number of which is Rad_ntp. Radiance contributions with total pathlength out of the range is not sampled.

Edit

Namelist mcarVis_nml_init

Vis_mrend

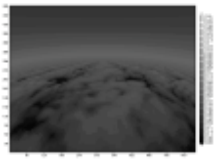
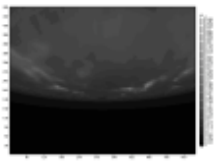
```
integer,  save :: Vis_mrend = 2
```

Method for volume rendering (0/1/2/3/4).

```
= 0 : integration of density along the ray (e.g. for calculating optical thickness)
  1 : as 1, but with attenuation (assuming that the source function is uniformly 1)
  2 : RTE-based solver, assuming horizontally-uniform source functions
  3 : as 2, but taking into accout 3-D distribution of 1st-order source functions (J1)
  4 : as 3, but with TMS correction for anisotropic scattering
```

Higher the value, more physically-correct, high-quality results will be obtained.

Note that the visualization mode is activated only when Wld_mtarget=3 in namelist mcarWld_nml_init. In addition, radiative source for the visualization mode is currently limited to solar and thermal sources: Local source is excluded.
Multiple scattering source functions are calculated by the Picard iteration for a single column with horizontally-averaged optical properties. The visualization modes with Vis_mrend >= 2 are all based on scaled RTE with delta-isotropic approximation (transport approximation).
Example results:

Wld_mtarget =	3	3	3	2
Vis_mrend =	2	3	4	(reference by
From above				
From above				

Vis_fatten

```
real(R_), save :: Vis_fatten = 1.0_R_
```

Attenuation factor for optical thickness along the line of sight. Should be 1 for physics-based rendering. This might be useful to get nice tone of visualized images, by increasing/decreasing physically-correct optical thickness.

Vis_epserr -technical parameter-

```
real(R_), save :: Vis_epserr = 0.001_R_
```

Convergence criterion for multiple scattering components, used only when Vis_mrend >= 2.

Vis_fpsmth -technical parameter-

```
real(R_), save :: Vis_fpsmth = 0.2_R_
```

Phase function smoothing fraction for relaxed TMS correction. When Vis_mrend = 4, TMS correction is used to reproduce highly anisotropic radiance angular distribution. The original TMS correction tends to overcorrect the radiance angular distribution when used with delta-isotropic approximation. Therefore, current MCARaTS visualization mode provides an easy method to relax the degree of overcorection. Phase function used for TMS correction is modified to

```
P_new = 1 - Vis_fpsmth + Vis_fpsmth * P_orig
```

instead of the original phase function P_orig.

Vis_nqhem -technical parameter-

```
integer,  save :: Vis_nqhem = 1
```

Number of streams (Gaussian quadrature points) in a hemisphere. Used for the Picard iteration for multiple scattering source function.

Vis_nqlay -technical parameter-

```
integer,  save :: Vis_nqlay = 5
```

Number of Gaussian quadrature points per layer, for optical thickness integration. This is used when the ray direction is nearly horizontal and/or when the atmospheric cells have very large vertical-to-horizontal aspect ratio. To reduce computational time, Gaussian quadrature will be used for integration of layer optical thickness, instead of cell-by-cell ray tracing.

Edit

Namelist mcarPho_nml_init

Pho_iso_SS

```
integer,  save :: Pho_iso_SS  = 1
```

The maximum scattering order, at which photons are allowed to move in the 3-D. Used only when the partially-three-dimensional (P3D) RT solver is selected. The P3D solver allows photons with lower-order of scattering to move in the 3-D. After a scattering or reflection event of the order of Pho_iso_SS, the photon packet never moves in horizontal direction, i.e., independent column approximation (ICA) is used for higher orders.
The solver selection is done when the user run the mcarats executables, by a command line argument.

Pho_iso_max -technical parameter-

```
integer,  save :: Pho_iso_max = 10000000
```

The maximum order of scattering for simulation. After the scattering of this order, tracing of photon packet will terminate. This might be useful for obtaining approximate results for optically thick, weakly-absorbing atmosphere. For physically-correct results, however, this parameter should be set at a very large value (about 10000 or larger). This parameter is also useful for obtaining results (Pho_iso_max = 1) with only single-scattering components, for example.

Pho_iso_tru -technical parameter-

```
integer,  save :: Pho_iso_tru = 0 ! min scattering order for truncation approximations
```

Maximum order of collision for non-approximated radiative transfer simulation. For

collision events with order less than or equal to Pho_iso_tru, no truncation approximation is used. The approximation can introduce (usually very small) bias. Thus this parameter might be useful for some purposes. For example, if the user need to get accurate separation of direct-beam and multiple scattering, then setting as Pho_iso_tru = 1 forces the direct beams non-approximated and then for multiple scattering the truncation approximations can be used.

Pho_wsml -technical parameter-

```
real(R_), save :: Pho_wsml   = 0.05_R_
```

Russian roulette weight limit for using local estimates. The local estimation is performed only for photons with large enough weight. For a photon with small weight, the Russian roulette is operated first, before calculating the phase function and ray tracing from the collision point to the detector. This weight threshold is independent of thresholds used in the local estimation modules.

Pho_wmin, Pho_wmax -technical parameter-

```
real(R_), save :: Pho_wmin   = 0.2_R_
real(R_), save :: Pho_wmax   = 3.0_R_
```

Minimum and maximum limits, respectively, for ideal weight of single photon packet. These limits avoid large dispersion in the photon weight. See also Pho_wfac.

Pho_wfac -technical parameter-

```
real(R_), save :: Pho_wfac   = 1.0_R_
```

Factor to modify the ideal weight of photon packet. The ideal weight is multiplied by Pho_wfac once every collision events. If the weight of photon packet is less than the ideal weight, then the Russian roulette method is applied, and photon population is reduced. Thus survived photon packets should have large enough weight compared to the ideal weight.

Pho_pfpeak -technical parameter-

```
real(R_), save :: Pho_pfpeak = 30.0_R_
```

Phase function peak threshold, used for adaptive truncation approximation. Default value is generally optimal.
The adaptive use of truncation approximations is a bit complicated. For every phase functions, multiple truncation approximations are prepared in the preprocess. An order of approximation is adaptively determined, depending on history of each photon trajectory. The approximation order is first initialized for the source emission depending on the isotropy of source emission angular distribution. The order usually increase with the order of scattering. However, actual determination of the approximation order is a bit complicated. The order is increased by one each scattering in a direction that corresponds to small enough value of phase function. If the phase function value is larger than the threshold Pho_pfpeak, the photon is considered as to be redirected to phase function peak direction. Such photons likely contribute to large anisotropy of radiance angular distribution, and high degrees of approximation is not suitable to

obtain high accuracy, so that the approximation degree is not changed at the scattering event.

Edit

Next / Return

添付ファイル

- img_tut_out1.png
- img_tut_out2.png
- img_vis_test2_mrend2.png
- img_vis_test2_mrend3.png
- img_vis_test2_mrend4.png
- img_vis_test2_ref.png
- img_vis_test3_mrend2.png
- img_vis_test3_mrend3.png
- img_vis_test3_mrend4.png
- img_vis_test3_ref.png

Next / Return

- Namelist mcarWld_nml_job
  - Wld_nplcf -technical parameter-
  - Wld_rmlcf -technical parameter-
  - Wld_dtaumin -technical parameter-
- Namelist mcarAtm_nml_job
  - Atm_idread
  - Atm_dx, Atm_dy
  - Atm_zgrd0
  - Atm_wkd0
  - Atm_tmp1d
  - Atm_ext1d
  - Atm_omg1d
  - Atm_apf1d
  - Atm_abs1d
  - Scaling optical properties
  - Atm_fext1d
  - Atm_fext3d
  - Atm_fabs1d
  - Atm_fabs3d
  - Atm_taumin -technical parameter-
  - Atm_fsupg -technical parameter-
  - Atm_fsupi -technical parameter-
  - Atm_nzzsup -technical parameter-
- Namelist mcarSfc_nml_job
  - Sfc_idread
  - Sfc_tmp
  - Sfc_mtype
  - Sfc_param
  - Sfc_nudsm, Sfc_nurpv, Sfc_nulsrt -technical parameter-
  - Sfc_nqpot -technical parameter-
  - Sfc_rrmax -technical parameter-
  - Sfc_rrexp -technical parameter-
- Namelist mcarSrc_nml_job
  - Src_mtype
  - Src_wlen
  - Src_dwlen
  - Src_flx
  - Src_qmax
  - Src_the, Src_phi
  - Src_mphi
  - Src_xpos, Src_ypos
  - Src_zloc
  - Src_apsize
- Namelist mcarRad_nml_job
  - Rad_mrproj
  - Rad_zloc
  - Rad_xpos, Rad_ypos
  - Rad_rmin0
  - Rad_rmax0
  - Rad_frmod
  - Rad_the, Rad_phi, Rad_psi
  - Rad_umax, Rad_vmax

- Rad_qmax
- Rad_apsize
- Rad_wfunc0
- Rad_zetamin -technical parameter-
- Rad_difr0, Rad_difr1 -technical parameter-
- What variables are actually used?

Edit

Namelist mcarWld_nml_job

Wld_nplcf -technical parameter-

```
integer,  save :: Wld_nplcf  = 0
```

Power exponent for local collision forcing (LCF), which forces collisions near the radiance detectors, by increasing extinction coefficients near the radiance detectors. The LCF is automatically deactivated if radiances are not target.
The LCF is effective when the 1st kind of radiances (Rad_mrkind = 1) are calculated. The formula of the 1st kind of radiance contribution has a term proportional to 1/r^2, where r is a distance between the collision point and radiance detector point. This type of local estimate has in general infinite variance. To reduce variance, LCF is useful. The power exponent should be 1 or 2 usually.
Scaled extinction coefficients (B_ext1) will be

```
B_ext1 = max(B_min, B_ext)
```

for original extinction coefficient B_ext, and the min extinction coefficient is

```
B_min = Wld_dtaumin / dz_lay * (zdif / Wld_rmlcf)**Wld_nplcf
```

where dz_lay is a layer geometrical thickness and zdif is height (z-coordinate) difference from the camera location. See also Wld_rmlcf and Wld_dtaumin.

Wld_rmlcf -technical parameter-

```
real(R_), save :: Wld_rmlcf  = 100.0_R_
```

Scale length (m) for Local Collision-Forcing (LCF), which forces artificial collisions at locations near the radiance detectors. See also Wld_nplcf.

Wld_dtaumin -technical parameter-

```
real(R_), save :: Wld_dtaumin = 0.0_R_
```

This has different effects on extinction coefficient scaling, depending on target radiative quantities.
When fluxes and heating rates are target or the local collision forcing (LCF) is not used (Wld_nplcf = 0), Wld_dtaumin is a minimum layer optical thickness for collision forcing. If optical thickness of one layer is less than this threshold, then the collision-forcing method is used in that layer, and the scaled value of the optical thickness is set to Wld_dtaumin. Thus collision events are forced to occur with prescribed probability even in very optically thin layer.
When radiances are target and the LCF is used, then Wld_dtaumin is used for the LCF. See the description of Wld_nplcf for how each parameter is used in the code.

Edit

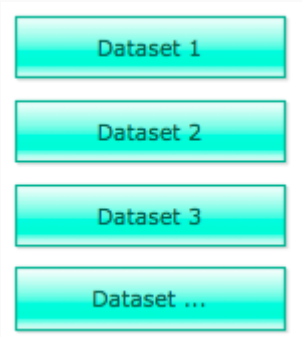
Namelist mcarAtm_nml_job

Several 1-D vertical profiles of some variables can be specified in this namelist. Data for 3-D distribution of properties for horizontally-inhomogneous layers are usually very large. Therefore, they should be read in from the external data file. The 1-D and 3-D profile data are mixed and used in the code. Please work with care for how they are mixed, which could be different by variable. See here (in Chapter 3) for how homogeneous and inhomogeneous layers are defined.

Atm_idread

```
integer,  save :: Atm_idread = 1
```

This controls how to import atmospheric dataset for the current job. Definition depends on the file format: binary or text. In both cases, if Atm_idread = 0, then namelist input for Atm_ext1d, Atm_omg1d, Atm_apf1d, and Atm_abs1d are all neglected. For the first job, Atm_idread should be > 0.
This variable also controls what dataset is read in from the external data file, Atm_inpfile, for the current job. Note that it is assumed that multiple datasets are possibly stored in the file as in the figure below.



Binary format : Atm_mfmt = 1
Variable Atm_idread is location index of data set to be read, in the 3-D atmospheric data file. One of data sets in the file can be chosen for each simulation job. The user can specify explicitly the data set index by this parameter or leave it as default value, which is 1 for the first job, and 0 for the subsequent jobs.
If Atm_idread = 0, then no new data set is read in for the current job, and atmospheric properties are the same as previous job. This is possible for second or subsequent job.
As described here, there are many ways to specify explicitly or implicitly the data set index for a job. For example,

- As default for all jobs

Job index	1	2	3	4	5	6
Atm_idread	-	-	-	-	-	-
Index of used data set	1	1	1	1	1	1

- Common single data set for all jobs

Job index	1	2	3	4	5	6
Atm_idread	5	-	-	-	-	-
Index of used data set	5	5	5	5	5	5

- To give explicitly all the indexes

Job index	1	2	3	4	5	6
Atm_idread	5	2	1	3	5	4
Index of used data set	5	2	1	3	5	4

- A combined aproach

Job index	1	2	3	4	5	6
Atm_idread	5	-	-	3	0	4
Index of used data set	5	5	5	3	3	4

where "-" denotes that no valye for Atm_idread is given in the namelist.
Text format : Atm_mfmt = 0
Variable Atm_idread is on/off flag for data import. If 0, no data is read in for the current job, and previously imported data are used again. If 1, then a new data set is read in from the file. For text format file, there is no possibility of random access as in binary format. Thus data sets are read in sequentially for each job.

Atm_dx, Atm_dy

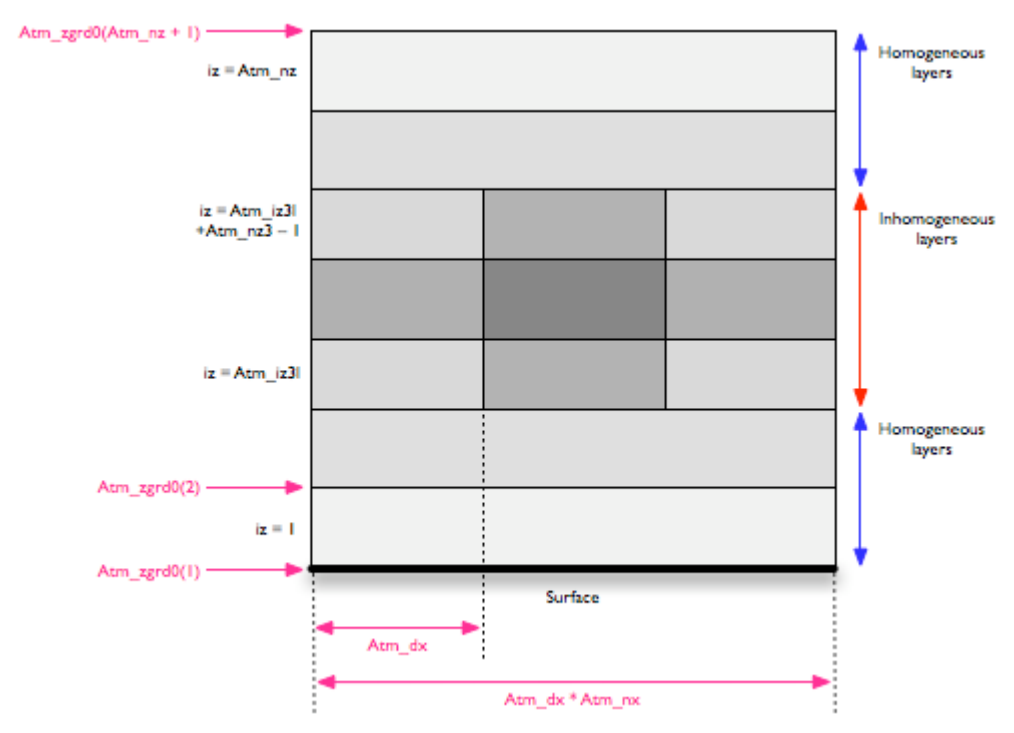
```
real(R_), save :: Atm_dx = 1.0e+4_R_ ! X size of cell
real(R_), save :: Atm_dy = 1.0e+4_R_ ! Y size of cell
```

Grid spacing (cell length and width) along x and y-axes, respectively.

Atm_zgrd0

```
real(R_), save :: Atm_zgrd0(1:KNZ+1) = (/(10.0_R_*i, i = 1, KNZ+1)/)
```

Z-coordinate values (m) of layer boundaries. The bottom and top of a layer with index iz correspond to z=Atm_zgrd0(iz) and Atm_zgrd0(iz+1), respectively. The BOA (surface) corresponds to Atm_zgrd0(1), and TOA Atm_zgrd0(Atm_nz+1).
The following illustration shows the definitions of several geometrical parameters.



Atm_wkd0

```
real(R_), save :: Atm_wkd0(KNKD) = (/1.0_R_, (0.0_R_, i = 1, KNKD - 1)/) ! k-distribution weights
```

Weight coefficients of the correlated k-distribution. Give Atm_wkd0(1)=1 for monochromatic computation.

Atm_tmp1d

```
real(R_), save :: Atm_tmp1d(1:KNZ+1) = (/(300.0_R_,  i = 1, KNZ+1)/)
```

Horizontally-averaged atmospheric temperatures [K] defined

- at layers [1:Atm_nz] if Atm_mtprof=0, or
- at layer interfaces [1:Atm_nz+1] if Atm_mtprof=1.

Used for thermal emission when Src_mtype = 2 or 3.
When Atm_mtprof=1, an interface of iz=1 corresponds to BOA, and that of iz=Atm_nz+1 corresponds to TOA. Note that Atm_tmp1d is atmospheric temperature even when iz=0 (BOA). The surface temperature should be given by different variables. For horizontally inhomogeneous layer, data of perturbation in temperature could be specified in external data file.

Atm_ext1d

```
real(R_), save :: Atm_ext1d(KNZ, KNP1D)
```

Extinction coefficients (/m) for each layer and each scattering component, one of which can be gas, aerosol, hydrometeor, or their mixture. Extinction coefficient Atm_ext1d(iz, ip1d) corresponds to a layer iz and a component ip1d. Note that optical properties specified by Atm_*1d variables are assumed be horizontally homogeneous in every atmospheric layers.

Atm_omg1d

```
real(R_), save :: Atm_omg1d(KNZ, KNP1D)
```

As Atm_exp1d, but for single scattering albedo.

Atm_apf1d

```
real(R_), save :: Atm_apf1d(KNZ, KNP1D)
```

As Atm_exp1d, but for phase function specification parameters. Phase functions are specified in the following way:

```
Atm_apf1d <= -2      : isotropic scattering phase function
-2 < Atm_apf1d <= -1 : Rayleigh scattering phase function
-1 < Atm_apf1d < 1   : Henyey-Greenstein phase function with asymmetry parameter of Atm_apf1d
1 <= Atm_apf1d       : a tabulated phase function (given by the database file, see the namelist mcaSca_nml_init)
```

Note Atm_apf1d should be <= Sca_npf (# of phase functions read in from the file). Even when Atm_apf1d >= 1, this variable does not need to be integer. The phase function is assumed to follow an equation:

```
P = (1 - a) * P_tab(i) + a * P_tab(i+1)
```

where

```
i = int(Atm_apf1d)
a = Atm_apf1d - i
```

That is, the phase function is linearly interpolated between P_tab(i) and P_tab(i+1). Thus, for better accuracy, the parameter Atm_apf1d should be given, taking into account a relationship between the phase function and a physical property of particles. For example, note that the phase function is linear to the inverse of the effective radius (Re), as a first order approximation:

```
P ~ c + d / Re
```

Atm_abs1d

```
real(R_), save :: Atm_abs1d(KNZ, KNKD)
```

Horizontally-averaged absorption coefficients (/m) of purely-absorbing component, which can be gaseous mixture for instance. The correlated k-distribution is assumed. An element Atm_abs1d(iz, ikd) corresponds to a layer iz and a k-distribution index ikd. Note perturbations of the absorption coefficients in the horizontally inhomogeneous layers can be given by the data file (Atm_inpfile).

Scaling optical properties

Input extinction and absorption coefficients can be easily scaled by user-specified factors, as described below. The scaling is performed immediately after reading in the data (from the namelist file or from 3-D atmospheric data file). Variables, Atm_fext1d, Atm_fext3d, Atm_fabs1d, Atm_fabs3d, have been introduced in the namelist. These are useful for simple adjustments of input data. For example, someone could try two simulations easily: one for cloudy sky with 3-D cloud extinction, the other without the cloud (e. g., by setting Atm_fext3d = 0 for cloud).
Note that these parameters are effective only when Atm_idread >= 1. When Atm_idread = 0, no new dataset is read in, and no data scaling is applied.

Atm_fext1d

```
real(R_), save :: Atm_fext1d(KNP1D) = (/(1.0_R_, i = 1, KNP1D)/)
```

Scaling factor for Atm_ext1d(ip1d), where ip1d is component index. This is neglected when Atm_idread = 0: a new data set is not read in for the current job.
This provide a simple way to modify the original extinction coefficients given by Atm_ext1d. The data are modified just after reading in the namselist data, in the following way:

```
B_ext(iz, ip1d) = Atm_ext1d(iz, ip1d) * Atm_fext1d(ip1d)
```

for all iz = [1, Atm_nz].
Note the values of Atm_ext1d are untouched by this scaling and modified data are saved in a different variable array. Therefore, values of Atm_ext1d will be saved in the subsequent jobs.

Atm_fext3d

```
real(R_), save :: Atm_fext3d(KNP3D) = (/(1.0_R_, i = 1, KNP3D)/)
```

As Atm_fext1d, but for modifying Atm_ext3d, 3-D spatial distribution of extinction coefficients. This is neglected when Atm_idread = 0: a new data set is not read in for the current job.
The 3-D data are read in from the external data file (Atm_inpfile). The data are modified just after reading in the 3-D data, in the following way:

```
Atm_ext3d(ix, iy, iz3, ip3d) = Atm_ext3d(ix, iy, iz3, ip3d) * Atm_fext3d(ip3d)
```

for all iz3 = [1, Atm_nz3].
Note that a new data set is not always read in for each job and that the modification by Atm_fext3d is applied only when a new data set is read in. See also Atm_idread for when a new data set is read in for a job.

Atm_fabs1d

```
real(R_), save :: Atm_fabs1d = 1.0_R_
```

Scaling factor for Atm_abs1d. This is neglected when Atm_idread = 0: a new data set is not read in for the current job.
This provide a simple way to modify the original absorption coefficients given by Atm_abs1d. The data are modified just after reading in the namselist data, in the following way:

```
B_abs(iz, ikd) = Atm_absld(iz, ikd) * Atm_fabsld
```

for all iz = [1, Atm_nz] and all ikd = [1, Atm_nkd].
Note the values of Atm_abs1d are untouched by this scaling and modified data are saved in a different variable array. Therefore, values of Atm_abs1d will be saved in the subsequent jobs.

Atm_fabs3d

```
real(R_), save :: Atm_fabs3d = 1.0_R_
```

As Atm_fabs1d, but for modifying Atm_abs3d, 3-D spatial distribution of absorption coefficients. This is neglected when Atm_idread = 0: a new data set is not read in for the current job.
The 3-D data are read in from the external data file (Atm_inpfile). The data are modified just after reading in the 3-D data, in the following way:

```
Atm_abs3d(ix, iy, iz3, ikd) = Atm_abs3d(ix, iy, iz3, ikd) * Atm_fabs3d
```

for all iz3 = [1, Atm_nz3] and all ikd = [1, Atm_nkd].
Note that a new data set is not always read in for each job and that the modification by Atm_fabs3d is applied only when a new data set is read in. See also Atm_idread for when a new data set is read in for a job.

Atm_taumin -technical parameter-

```
real(R_), save :: Atm_taumin = 2.0_R_
```

Minimum column optical thickness used for collision forcing. If actual column optical thickness is less than this threshold, then the collision forcing method is used, and the scaled value of the total optical thickness is set to Atm_taumin. Recommended value is between 1 and 10.

Atm_fsupg -technical parameter-

```
real(R_), save :: Atm_fsupg  = 2.0_R_
```

Geometrical threshold parameter for automatic, adaptive super-cell formation for maximum cross section method. In the preprocess, super-cells are formed by merging cells until one of conditions are met. Cells are merged until super-cell size (m) is larger, on average, than

```
Atm_fsupg / E_max
```

where E_max is the maximum extinction coefficient (/m).
This is one of three conditions for adaptive super-cell formation. See also Atm_fsupi and Atm_nzzsup.

Atm_fsupi -technical parameter-

```
real(R_), save :: Atm_fsupi  = 1.3_R_
```

Inhomogeneity threshold parameter for automatic, adaptive super-cell formation for maximum cross section method. Cells are merged when inhomogeneity of extinction coefficients within the super-cell is small enough, on average. If ratio of the maximum extinction coefficient (E_max) to super-cell average extinction coefficient (E_ave) is larger than Atm_fsupi, then cell merging will be stopped:

```
E_max / E_ave > Atm_fsupi
```

This is one of three conditions for adaptive super-cell formation. See also Atm_fsupg and Atm_nzzsup.

Atm_nzzsup -technical parameter-

```
integer,  save :: Atm_nzzsup = 50
```

Possible max number of atmospheric layers merged into a single super-cell, in which the maximum cross section method is used.

Edit

Namelist mcarSfc_nml_job

Sfc_idread

```
integer,  save :: Sfc_idread = 1
```

Location index of data set to be read, in the surface data file (Sfc_inpfile). Data set specification method is the same as Atm_idread. See Atm_idread for details.

Sfc_tmp

```
real(R_), save :: Sfc_tmp = 300.0_R_
```

Average surface temperature (K). Used for calculation of thermal emission when Src_mtype = 2 or 3. 2-D temperature perturbation can be specified by the external data file (Sfc_inpfile).

Sfc_mtype

```
integer,  save :: Sfc_mtype = 1
```

Type index of globally constant surface BRDF. Used only when Sfc_inpfile = '', i.e. BRDF data are not given by the file, Sfc_inpfile. BRDF is assumed as follows:

```
0: black
1: Lambertian
2: Flx_diffuse-specular mixture (DSM) model
3: Rahman-Pinty-Verstraete (RPV) model
4: Li-Sparse-Ross-Thick (LSRT) model
```

Sfc_param

```
real(R_), save :: Sfc_param(Sfc_NPAR) = (/1.0_R_, 0.0_R_, 0.0_R_, 0.0_R_, 0.0_R_/)
```

BRDF parameters for globally constant surface properties. Used only when Sfc_inpfile = '', i.e. BRDF data are not given by the file, Sfc_inpfile. Definitions of Sfc_param(ipar) for respective ipar differs by the BRDF type, Sfc_mtype.

Sfc_mtype = 0 (black): No specification is required for Sfc_param, which is dummy parameter.
Sfc_mtype = 1 (Lambertian):

- Sfc_param(ipar=1): albedo

Sfc_mtype = 2 (DSM): The surface reflection is modeled by a microscopical mixture of diffuse and specular reflection.

- Sfc_param(ipar=1): diffuse albedo (A_d)
- Sfc_param(ipar=2): fraction of Flx_diffuse reflectors (f_d); Fraction of specular reflection is 1 - Sfc_param(ipar=2)
- Sfc_param(ipar=3): real part of refractive index of underlying material (e.g., water); Should be given as positive.
- Sfc_param(ipar=4): imaginary part of refractive index of underlying material (e.g., water); Should be given as positive.
- Sfc_param(ipar=5): variance of micro-scopic surface slope (tangent of the inclination angle). Give zero for flat surface. This is used for specular reflection. According to Cox and Munk's (1954) parameterization for ocean surface,

```
Sfc_param(ipar=5) = 0.00512 * V + 0.003.
```

where V is wind velocity (m/s) at 10 m above the ocean surface.
The DSM model albedo (A) and BRDF (R) follow

```
A = f_d * A_d + (1 - f_d) * A_s,
R = f_d * R_d + (1 - f_d) * R_s,
pi * R_d = cosQ * A_d
```

where A_s and R_s are albedo and BRDF for specular reflection.

Sfc_mtype = 3 (RPV): Rahman-Pinty-Verstraete (RPV) BRDF model

- Sfc_param(ipar=1): the first parameter, alpha
- Sfc_param(ipar=2): the exponent parameter, k
- Sfc_param(ipar=3): the asymmetry parameter, THETA
- Sfc_param(ipar=4): delta

The RPV BRDF, R, is given as

```
pi * R = alpha * [u0 * u1 / (u0 + 1) / (u1 + 1)]^(k - 1) * HG(THETA) * F(G)
```

where pi=3.141592653, u0 and u1 are respectively cosines of incidence and reflection vectors, and HG is the Heyney-Greenstein function. The function F is as follows:

```
F(G) = 1 + (1 + G)/(delta + G),
```

where G is a geometrical parameter.

Sfc_mtype = 4 (LSRT): Li-Sparse-Ross-Thick (LSRT) BRDF model

- Sfc_param(ipar=1): "k_L", weight of Lambertian reflection
- Sfc_param(ipar=2): "k_g", weight of geometrical optics reflection
- Sfc_param(ipar=3): "k_v", weight of volume scattering

The LSRT BRDF, R, is given by

```
pi * R = k_L * pi + k_g * K_geo + k_v * K_vol
```

where K_geo and K_vol are geometrical and volume scattering kernel functions.

Sfc_nudsm, Sfc_nurpv, Sfc_nulsrt -technical parameter-

```
integer,   save :: Sfc_nudsm  = 14
integer,   save :: Sfc_nurpv  = 8
integer,   save :: Sfc_nulsrt = 14
```

Number of solar zenith angles for the look-up tables (LUTs) for DSM, RPV, and LSRT BRDF models, respectively. The LUTs contains parameters for albedo calculation and determination of reflection direction. In simulations, the parameters used are interpolated from the LUT. Thus larger Sfc_nudsm value achieves higher accuracy but requires larger memory for the LUTs.

Sfc_nqpot -technical parameter-

```
integer,   save :: Sfc_nqpot  = 24
```

Number of quadrature points for surface albedo pre-calculations for the BRDF models, DSM and RPV models. Used when creating the LUTs for albedo, which is calculated by integration of the BRDF over the hemisphere. Larger quadrature points can achieve accurate results but may take longer CPU time. Usually 10 is enough to achieve high accuracy as 0.5% (relative unit).

Sfc_rrmax -technical parameter-

```
real(R_), save :: Sfc_rrmax  = 5.0_R_
```

Parameter for determination of random reflection direction by the BRDF models, DSM, RPV, and LSRT models. The rejection method is used for sampling. Smaller value needs less computer time, but the determined reflection direction is more approximate. If Sfc_rrmax is very large (>1.0e+8), then no such an approximation is used, and the simulation may require nearly infinite CPU time! Should be larger than or equally to 1. Default value is generally optimal.

Sfc_rrexp -technical parameter-

```
real(R_), save :: Sfc_rrexp  = 0.5_R_
```

Parameter for determination of random reflection direction by the BRDF models, DSM and RPV models. Smaller value needs less computer time but the determined reflection direction is more approximate. If Sfc_rrexp=1, then no such an approximation is used, and the simulation may require nearly infinite CPU time! Should be between 0 and 1. Default value is generally optimal.

Edit

Namelist mcarSrc_nml_job

Src_mtype

```
integer,   save :: Src_mtype(KNSRC) = (/(1,  i = 1,  KNSRC)/)
```

Flag for source type (0,1,2,3). Defines external and/or internal sources, as follows:

```
0: localized sources (the location, direction, and angular width should be specified)
1: solar sources (incidence from the top of atmosphere)
2: solar plus thermal emission sources
3: thermal emission sources
```

Src_wlen

```
real(R_), save :: Src_wlen(KNSRC)   = (/(10.0_R_,   i = 1,  KNSRC)/)
```

Wavelength (micrometer) of light for thermal emission. This is used only when Src_mtype = 2 or 3. If the wavelength band width (Src_dwlen) is not zero, then Src_wlen is the lower limit of wavelength. See also Src_dwlen.

Src_dwlen

```
real(R_), save :: Src_dwlen(KNSRC) = (/0.1_R_,    i = 1, KNSRC)/)
```

Wavelength band width (micrometer). Zero is acceptable. When Src_dwlen=0, the simulation is completely monochromatic, and output quantities are spectral ones. Otherwise (Src_dwlen > 0), output quantities are spectrally averaged ones, and spectrally averaged thermal emission are calculated when Src_mtype = 2 or 3.
Note on output quantities and physical units:
a) Src_dwlen=0: spectral, monochromatic calculations, as the followings

- area-averaged spectral flux densities (e.g., W/m^2/micrometer),
- volume-averaged spectral heating rates (e.g., convergences, W/m^3/micrometer),
- spectral radiances (e.g., W/m^2/steradian/micrometer),

b) Src_dwlen > 0: spectrally-averaged calculations; Units are the same as above.

Src_flx

```
real(R_), save :: Src_flx(KNSRC)   = (/1.0_R_,    i = 1, KNSRC)/)
```

Spectral or spectrally-averaged source flux density or power.

- When the solar source is present (Src_mtype = 1 or 2), Src_flx is the solar source spectral irradiance (W/m^2/micrometer) at the top of atmosphere. The Src_flx is defined as integrated on the plane normal to the center direction of the source emission. The top-of-atmosphere solar source irradiance on the horizontal plane should be, for collimated solar incidence (Src_qmax=0),

```
Src_flx * abs(cos(180 - Src_the))
```

- When the source is localized (Src_mtype = 0), Src_flx is the emitted spectral power (W/micrometer).

Even if Src_dwlen > 0, Src_flx should be spectrally-averaged quantity, averaged over the wavelength band. Then spectrally integrated solar source irradiance should be Src_dwlen*Src_flx, for example.

Src_qmax

```
real(R_), save :: Src_qmax(KNSRC)  = (/0.0_R_,    i = 1, KNSRC)/)
```

Full cone angle (degrees) that determines angular width of solar or localized source, used only when Src_mtype = 0, 1 or 2. This should be 0.533133 (degree) for solar incidence at an annual-mean state of the sun-earth system. Give zero for cases in which sources are negligibly small or infinitely far.

Src_the, Src_phi

```
real(R_), save :: Src_the(KNSRC)   = (/120.0_R_,  i = 1, KNSRC)/) ! zenith angle
real(R_), save :: Src_phi(KNSRC)   = (/0.0_R_,    i = 1, KNSRC)/) ! azimuth angle
```

Zenith and azimuth angles (degrees), respectively, of the source emission direction vector. Used only when Src_mtype = 0, 1 or 2. Note the source direction is defined as transport direction of source light. If the source is solar light, then the transport direction corresponds to the sun-to-surface vector. Thus, the solar zenith angle should be (180 - Src_the).

Src_mphi

```
integer,  save :: Src_mphi(KNSRC)  = (/0,   i = 1, KNSRC)/)
```

Flag for random azimuth of source emission (0:no, 1:yes). If Src_mphi = 0, then the azimuth angles of the source vectors are set as Src_phi. Else if Src_mphi = 1, then the azimuth angles are set as random between 0 to 360 degrees.

Src_xpos, Src_ypos

```
real(R_), save :: Src_xpos(KNSRC) = (/0.5_R_,  i = 1, KNSRC)/)  ! X relative position
real(R_), save :: Src_ypos(KNSRC) = (/0.5_R_,  i = 1, KNSRC)/)  ! Y relative position
```

Coordinates that specifies the location of the localized source, used when Src_mtype=0.

Src_zloc

```
real(R_), save :: Src_zloc(KNSRC)  = (/0.0_R_,  i = 1, KNSRC)/)
```

Z-coordinate that specifies the location of the localized source, used when Src_mtype=0.

Src_apsize

```
  real(R_), save :: Src_apsize(KNSRC) = (/(0.0_R_,    i = 1, KNSRC)/)
```

Aperture size, diameter (m) of the local source surface (or the surface of the outmost lens of the artificial source), when Src_mtype = 0. The localized source should be very small compared to the model domain, but non-zero finite size is allowed. If Src_apsize = 0, then it means the source is from a point. Otherwise, the source surface is assumed to be circular. Emission surface is assumed to be on a plane perpendicular to the emission center direction (with Src_the & Src_phi).

Edit

Namelist mcarRad_nml_job

Rad_mrproj

```
  integer,  save :: Rad_mrproj = 0
```

Flag for angular weighting, when radiances are averaged over finite solid angle (Rad_mrkind = 1 or 3). Options are as follows:

```
= 0 : w = 1, results are radiances simply averaged over solid angle
= 1 : w = cosQ for Q = angle from the camera center direction, results are weighted average
```

where w is a weighting function. Output radiance will be

```
R_out = integral{R(s) * w(s) * dS} / integral{w(s) * dS}
```

where R(s) is radiance in a direction s, and S is solid angle within the sensor FOV.
Here are two eamples. When FOV is a hemisphere (Rad_umax = 90 deg.),

```
Rad_mrproj = 0: R_out = hemispherical-mean radiance, and R_out * (2*pi) = actinic flux density (spheradiance)
Rad_mrproj = 1: R_out * pi = irradiance (flux density)
```

Rad_zloc

```
  real(R_), save :: Rad_zloc(KNRAD) = (/(  0.0_R_, i = 1, KNRAD)/)
```

Z coordinate, altitude (m) of the sensor location.
The pixel mapping for radiance is performed at the sensor location. Note that this differs from the method that is used for map projection of satellite observation image pixels, which are mapped at the ground level (not at the satellite location).

Rad_xpos, Rad_ypos

```
  real(R_), save :: Rad_xpos(KNRAD) = (/(  0.5_R_, i = 1, KNRAD)/) ! X relative position
  real(R_), save :: Rad_ypos(KNRAD) = (/(  0.5_R_, i = 1, KNRAD)/) ! Y relative position
```

Relative horizontal location along x and y-axes, respectively, for the sensor. X and Y coordinates at the sensor location are respectively

```
X_cam = Rad_xpos * Atm_dx * Atm_nx
Y_cam = Rad_ypos * Atm_dy * Atm_ny
```

Thus, Rad_xpos=0 corresponds to the X coordinate of 0, and Rad_xpos=1 corresponds to the uppermost X coordinate of the domain. If Rad_xpos or Rad_ypos is out of a range [0, 1], then they are reset in the code under the cyclic boundary condition. For example, Rad_xpos = 2.345 will be reset as 0.345, and -2.345 will be reset as 0.655.

Rad_rmin0

```
  real(R_), save :: Rad_rmin0(KNRAD) = (/(1.0e-17_R_, i = 1, KNRAD)/)
```

Minimum distance (m) between emission point and the sensor. Used only for Rad_mrkind = 1 or 2. Emission contributions from source emission, scattering, and reflection are all sampled for calculating radiances. A view region is defined for each sensor, and the distance R between the emission point and the sensor should be

```
Rad_rmin0 < R < Rad_rmax0
```

Emissions from out of the view region are all excluded from sampling. Transmittance between the line of sight segment for 0 < R < Rad_rmin0 is artificially set at 1.
If the first kind of radiances are calculated (Rad_mrkind = 1), simulated images can be very noisy when Rad_rmin0 is too small and scattering particles are present near the camera. This is because contribution function is proportional to 1/R^2. For that reason, emission near the camera should be excluded from sampling.
Note current code has an intelligent way to automatically set the view region: Rad_rmin0, Rad_rmax0, and Rad_frmod are automatically set taking into account the domain size, sensor location, and sensor FOV. Therefore, the user can leave the default

setting as is or alternatively set an explicit view region.

Rad_rmax0

```
real(R_), save :: Rad_rmax0(KNRAD) = (/(1.0e+17_R_, i = 1, KNRAD)/)
```

Maximum distance (m) between emission point and the sensor. Used only for Rad_mrkind = 1 or 2. This limits distance between emission/scattering point and the sensor. For simulation of physically-correct results, this limit should be infinity. However, such a simulation could be highly time consuming. See also Rad_rmin0.

Rad_frmod

```
real(R_), save :: Rad_frmod = 0.3_R_
```

Factor for mode distance between emission point and the sensor. Used only when the first kind of radiances are calculated (Rad_mrkind = 1). The mode distance (R_mod) will be

```
R_mod = sqrt( Rad_rmin0**2 + Rad_frmod * (Rad_rmax0**2 - Rad_rmin0**2) )
```

Contribution function for the first kind of radiance is proportional to 1/R^2. This mode distance is used for variance reduction. Emission contribution from a point far enough from the sensor, R > R_mod, is scaled by the Russian roulette:

```
Zeta_new = Zeta_old * (R / R_mod), if Xi < R_mod / R
Zeta_new = 0, otherwise
```

where Xi is a random number. This reduces populations of sampling from points far from the sensor, reducing calculation time.

Rad_the, Rad_phi, Rad_psi

```
real(R_), save :: Rad_phi(KNRAD) = (/(0.0_R_,    i = 1, KNRAD)/) ! phi,   angle around Z0
real(R_), save :: Rad_the(KNRAD) = (/(0.0_R_,    i = 1, KNRAD)/) ! theta, angle around Y1
real(R_), save :: Rad_psi(KNRAD) = (/(270.0_R_, i = 1, KNRAD)/) ! psi,   angle around Z2
```

Rotation angles (degrees) for the camera (sensor) coordinates, which are defined by rotating the original X-Y-Z Cartesian coordinates, following Z-Y-Z rotation rule. Values can be in a range (-Inf, +Inf). Three rotations are operated in the correct order:

```
1: rotation about Z0 (Z-axis in (X0,Y0,Z0) world coordinates) by Rad_phi
2: rotation about Y1 (Y-axis in (X1,Y1,Z1) coordinates) by Rad_theta
3: rotation about Z2 (Z-axis in (X2,Y2,Z2) coordinates) by Rad_psi
```

Finally camera coordinates are (Xc,Yc,Zc). Angle between Zc axis direction and the original Z0 axis direction should be Rad_theta, and azimuth angle of Zc axis direction in the original world coordinate system (X0,Y0,Z0) should be Rad_phi.
Therefore, Rad_theta and Rad_phi denotes the zenith and azimuth angles, respectively, in degree, of the center direction of sensor FOV. The direction is here defined as reverse of light transport, i.e. camera to target direction.
Rad_psi is dummy when Rad_mrkind = 2, i.e., sensor FOV is infinitesimal. Rad_psi should be appropriately set when Rad_mrkind = 1 or 3, because angular pixel mapping requires the rotation angle Rad_psi.

Rad_umax, Rad_vmax

```
real(R_), save :: Rad_umax(KNRAD) = (/(180.0_R_, i = 1, KNRAD)/) ! max angle along U-direction
real(R_), save :: Rad_vmax(KNRAD) = (/(180.0_R_, i = 1, KNRAD)/) ! max angle along V-direction
```

Maximum angles (degrees) of sensor FOV along camera image axes. Used only when Rad_mrkind = 1 or 3. Both should be <= 180 degrees. When Rad_mpmap = 2, Rad_vmax is not used.
Note pixel mapping method differs by Rad_mpmap. The image axes are here defined by U and V, and Rad_umax and Rad_vmax denote the maximum angles along U and V, respectively. U and V are closely related to the camera coordinates (Xc,Yc,Zc). See also the three angles (Rad_the, Rad_phi, Rad_psi) for how the camera coordinates are defined.
Rad_mpmap = 1 (rectangular mapping)

```
U = theta * cos(phi) + Rad_umax / 2, in [0, Rad_umax]
V = theta * sin(phi) + Rad_vmax / 2, in [0, Rad_vmax]
```

Rad_mpmap = 2 (polar mapping)

```
U = theta * 2, in [0, Rad_umax]
V = phi,       in [0, 360]
```

where theta and phi are the polar and azimuth angles of the line of sight (LOS) direction in the camera coordinate system. That is theta is the angle between Z_c axis direction and the LOS direction. See also Rad_qmax and a page for example applications.

Rad_qmax

```
real(R_), save :: Rad_qmax(KNRAD) = (/(180.0_R_, i = 1, KNRAD)/)
```

Maximum angle (degrees), full cone angle of the field of view (FOV) of the sensor. Should be <= 180 degrees. If Rad_qmax = 180 degrees, the the FOV corresponds to a hemisphere, the normal of which corresponds to the direction with Rad_the and Rad_phi.

Rad_apsize

```
real(R_), save :: Rad_apsize(KNRAD) = (/(0.0_R_, i = 1, KNRAD)/)
```

Aperture size, diameter (m) of the sensor surface (or the surface of the outmost lens of the sensor). Used only when the first kind of radiances are calculated (Rad_mrkind = 1).
The sensor size should be very small compared to the model domain, but non-zero finite size is allowed. If Rad_apsize = 0, then it means an infinitesimally small sensor. Otherwise, the sensor surface is assumed to be circular. Sensor surface is assumed to be on a plane perpendicular to the FOV center direction.

Rad_wfunc0

```
real(R_), save :: Rad_wfunc0(KNZ, KNWF)
```

Weighting functions used when column averaged pathlengths are calculated (Rad_mplen=2). Unit is arbitrary. An element Rad_wfunc0(iz, iwf) corresponds to a layer index iz and a weighting function index iwf. The weighting function is used in the following formula: air mass factor (AMF),

```
AMF(iwf) = sum{ L(iz) * Rad_wfunc0(iz, iwf) } / sum{ Rad_wfunc0(iz, iwf) } / Dz(iz)
```

where the sum is for all layers, iz = [1, Atm_nz], and L is pathlength and Dz is layer depth.
Multiple weighting functions are useful for calculating multiple air mass factors for multiple gaseous species with different vertical profiles.

Rad_zetamin -technical parameter-

```
real(R_), save :: Rad_zetamin = 0.3_R_
```

Threshold for each radiance contribution function sampled by the local estimation method. Radiance contribution from each emission/scattering/reflection event is calculated as the normalized angular distribution function multiplied by transmittance between the event point and the detector.

Rad_difr0, Rad_difr1 -technical parameter-

```
real(R_), save :: Rad_difr0 = 30.0_R_
real(R_), save :: Rad_difr1 = 0.01_R_
```

Artificial diffusion coefficients for radiances. Default values are generally optimal.

What variables are actually used?

Different sets of variables are used depending on the kind of radiance (Rad_mrkind). The following table summarizes which variables are used for a given Rad_mrkind.
Here, "y" denotes that the variable is used, and "-" not used.

Rad_mrkind	1	2	3	Only when	Rad_mrkind	1	2	3	Only when
Rad_zloc	y	y	y	*	Rad_rmin0	y	y	-	*
Rad_wfunc0	y	y	y	Rad_mplen>=1	Rad_rmax0	y	y	-	*
Rad_the	y	y	y	*	Rad_frmod	y	-	-	*
Rad_phi	y	y	y	*	Rad_apsize	y	-	-	*
Rad_psi	y	-	y	*	Rad_xpos	y	-	-	*
Rad_mrproj	y	-	y	*	Rad_ypos	y	-	-	*
Rad_umax	y	-	y	*					
Rad_vmax	y	-	y	Rad_mpmap=2					
Rad_qmax	y	-	y	*					

Edit

Next / Return

Next / Return

- External input file Sca_inpfile
  - ang
  - phs
  - na
  - Binary format
  - Text format
- External input file Atm_inpfile
  - Atm_tmpa3d
  - Atm_absg3d
  - Atm_extp3d, Atm_omgp3d
  - Atm_apfp3d
  - Data import procedure, common for binary and text formats
  - Binary format
  - Text format
- External input file Sfc_inpfile
  - Sfc_tmps2d
  - Sfc_jsfc2d
  - Sfc_psfc2d
  - Data import procedure, common for binary and text formats
  - Binary format
  - Text format

Edit

External input file Sca_inpfile

This file should contain data of scattering phase functions and can be in a text or binary format.
IMPORTANT NOTE : Binary format is highly recommended for faster data import and flexible random data accessibility. In the future versions, it is likely that only binary format files could be accepted.
Variables read in from this file are described below. Example data files can be found in the "examples" directory.
Numbers of elements should be specified in the namelist file:

```
Sca_nangi : max # of angles
Sca_npf   : # of phase functions
Sca_nskip : number of first data record lines to be skipped.
Sca_anci  : the number of ancillary data (definition differs between binary and text formats)
```

ang

```
real(R_), save, allocatable :: ang(:,:)     !(Sca_nangi, Sca_npf)
```

Scattering angles in degree for the phase function. Should be >= 0 and <= 180.

phs

```
real(R_), save, allocatable :: phs(:,:)     !(Sca_nangi, Sca_npf)
```

Phase functions. Relative values are acceptable, because the phase functions are automatically normalized in the code. See also Sca_knai.

na

```
integer,  save, allocatable :: na(:)        !(Sca_npf)
```

Number of angles for each phase function. Used only when the file is in text format, for which numbers of angles can be different for respective phase functions.

Binary format

Structure and format of the binary data file can be fully described by GrADS control file. An example is examples/les05_0045.sca.ctl:

```
DSET ^les05_0045.sca
*OPTIONS LITTLE_ENDIAN
```

```
*OPTIONS BIG_ENDIAN
OPTIONS template
TITLE les05_0045.sca
UNDEF 1.0e+37
XDEF       203 LINEAR 1 1
YDEF         1 LINEAR 1 1
ZDEF        60 LINEAR 1 1
TDEF         1 LINEAR 00:00Z00JAN0001 1YR
VARS         2
ang 1 99 Angles (deg.)
phs 60 99 Phase function
ENDVARS
```

The following Fortran code is used for reading data.

```
!####
  subroutine mcarUtl__binSca_read(iu, nangi, nanci, nphs, ang, phs, nskip, mbswap)
    !
    !* Read in phase functions from a binary file *

    integer,  intent(in) :: iu    ! file unit index
    integer,  intent(in) :: nangi ! # of angles
    integer,  intent(in) :: nanci ! # of ancillary data
    integer,  intent(in) :: nphs  ! # of phase functions
    real(R_), intent(out) :: ang(:) ! angles (degrees)
    real(R_), intent(out) :: phs(:,:) ! phase functions
    integer, intent(in), optional :: nskip  ! # of record lines to be skipped
    integer, intent(in), optional :: mbswap ! flag for byte swapping (0=no, 1=yes)
    real(R4_) :: wrk(nangi + nanci) !(AUTO)
    integer :: iphs, mswap, irec

    ! Initialize
    irec = 1
    mswap = 0
    if (present(nskip)) irec = nskip + 1
    if (present(mbswap) .and. mbswap == 1) mswap = 4

    ! Read in
    call bin_read_o1R4(iu, wrk, irec=irec, mswap=mswap)
    ang(1:nangi) = wrk(1:nangi)
    do iphs = 1, nphs
       irec = irec + 1
       call bin_read_o1R4(iu, wrk, irec=irec, mswap=mswap)
       phs(1:nangi, iphs) = wrk(1:nangi)
    end do

  end subroutine mcarUtl__binSca_read
```

Here, the subroutine arguments correspond to previously-described variables, as follows:

```
nangi --> Sca_nangi
nphs  --> Sca_npf
mbswap --> Wld_mbswap
```

In the above code, general purpose utility procedures (e.g., bin_read_o1R4) are used. See source codes, ./src/libhparx/libHparx_*.f90, for details.

Text format

The data block to be read is recognized after a line containing the following "signature":

```
%mdlphs
```

After the signature is found, the data are read in. The following Fortran code is used for reading data.

```
!####
  subroutine mcarUtl__txtSca_read(iu, nanci, na, ang, phs)
    !
    !* Get phase functions from a text-format file *

    integer,  intent(in)  :: iu       ! file unit index
    integer,  intent(in)  :: nanci    ! # of ancillary data blocks
    integer,  intent(out) :: na(:)    ! # of angles
    real(R_), intent(out) :: ang(:,:) ! angles (degrees)
    real(R_), intent(out) :: phs(:,:) ! phase functions
    !// Note: Number of ancillary data lines should be nanci*(np + 1).
```

```
    integer :: ia, ipf, nangi, npf, np, ip, naa, ianci, ierr

    ! Header
    nangi = size(ang, 1)
    npf   = size(ang, 2)

    ! Data
    ipf = 1
    loop_trial: do
       call skipToSign(iu, '%mdlphs', ierr)
       call err_read(ierr, iu, 'mcarUtl__txtSca_read: %mdlphs is not found.')
       read (iu, *, err=1, end=1)
       read (iu, *, err=1, end=1) naa
       call check_il('mcarUtl__txtSca_read: naa', naa, 1, nangi)
       read (iu, *, err=1, end=1) np
       do ianci = 1, nanci
          do ip = 1, np + 1
             read (iu, *, err=1, end=1)
          end do
       end do
       do ip = 1, np
          read (iu, *, err=1, end=1)
          na(ipf) = naa
          do ia = 1, na(ipf)
             read (iu, *, err=1, end=1) ang(ia, ipf), phs(ia, ipf)
          end do
          ipf = ipf + 1
          if (ipf > npf) exit loop_trial
       end do
    end do loop_trial

    return
1   call err_read(1, iu, 'mcarUtl__txtSca_read: ang & phs at ipf='//num2str_AN(ipf))

  end subroutine mcarUtl__txtSca_read
```

In the above code, general purpose utility procedures (e.g., bin_read_o1R4) are used. See source codes, ./src/libhparx/libHparx_*.f90, for details.

Edit

External input file Atm_inpfile

This file should contain data of atmospheric properties in horizontally-inhomogeneous layers and can be in a text or binary format. Note that the entire atmosphre is defined for all layers iz = [1:Atm_nz], but the horizontally-inhomogeneous layers are defined only for Atm_nz3 layers with iz = [Atm_iz3l : Atm_iz3l + Atm_nz3 - 1].
IMPORTANT NOTE : Binary format is highly recommended for faster data import and flexible random data accessibility. In the future versions, it is likely that only binary format files could be accepted.
Variables read in from this file are described below. Example data files can be found in the "examples" directory.
Numbers of elements should be specified in the namelist file:

```
Atm_nx, Atm_ny : # of volume elements along x and y axes
Atm_nz   : # of all layers
Atm_nz3  : # of horizontally-inhomogeneous layers
Atm_nkd  : # of k-distribution terms
Atm_np3d : # of scattering particle polydispersions in horizontally-inhomogeneous layers
```

Atm_tmpa3d

```
[real 3-D array, {((dat(ix, iy, iz3), ix=1, Atm_nx), iy=1, Atm_ny), iz3=1, Atm_nz3}] when Atm_mtprof=0
[real 3-D array, {((dat(ix, iy, iz3), ix=1, Atm_nx), iy=1, Atm_ny), iz3=1, Atm_nz3+1}] when Atm_mtprof=1
```

3-D distributions of perturbations of atmospheric temperature (K) at layers (Atm_mtprof=0) or layer interfaces (Atm_mtprof=1). Used for calculation of thermal emission when Src_mtype = 2 or 3.
When Atm_mtprof=1, an interface of index iz = Atm_iz3l (iz3=1) corresponds to the bottom boundary of the lowest inhomogeneous layer, and iz = Atm_iz3l + Atm_nz3 (iz3=Atm_nz3+1) corresponds to the top boundary of the highest inhomogeneous layer.
Planck function is assumed to vary exponentially with hight within each volume cell (i.e. log(B) is linear to altitude, where B is Planck function). When Atm_mtprof=0, temperature values at layer interfaces are automatically interpolated from the input profile.
Actual temperatures are represented as follows,

```
air_temp(ix, iy, iz) = Atm_tmp1d(iz) + Atm_tmpa3d(ix, iy, iz3), where iz = Atm_iz3l + iz3 - 1.
```

See also Atm_tmp1d and Atm_iz3l. Note that the above formula allows at least two methods:

1. Atm_tmp1d = 0, and Atm_tmpa3d for absolute temperature

Atm_tmp1d = horizontally-averaged temperature, and Atm_tmpa3d for perturbations

2.

Atm_absg3d

```
[real 4-D array, {(((dat(ix, iy, iz3, ikd), ix=1, Atm_nx), iy=1, Atm_ny), iz3=1, Atm_nz3), ikd=1, Atm_nkd}]
```

3-D distributions of perturbations of absorption coefficient (/m) for the correlated k-distribution. Actual absorption coefficients are represented as follows,

```
abs_coef(ix, iy, iz, ikd) = Atm_abs1d(iz, ikd) + Atm_absg3d(ix, iy, iz3, ikd)
```

where iz = Atm_iz3l + iz3 - 1. Note that the total coefficient may not be less than zero.

Atm_extp3d, Atm_omgp3d

```
[real 4-D array, {(((dat(ix, iy, iz3, ip3d), ix=1, Atm_nx), iy=1, Atm_ny), iz3=1, Atm_nz3), ip3d=1, Atm_np3d}]
```

Extinction coefficients (/m) and single scattering albedos, respectively, of particle components in horizontally-inhomogeneous layers with iz3=1 to Atm_nz3. A component can be a mixture of gases, aerosols and hydrometeors or a size-bin of their size distributions.

Atm_apfp3d

```
[integer 4-D array, {(((dat(ix, iy, iz3, ip3d), ix=1, Atm_nx), iy=1, Atm_ny), iz3=1, Atm_nz3), ip3d=1, Atm_np3d}]
```

Phase function specification parameter for each scattering component in horizontally inhomogeneous layers. Phase functions are specified in the same way as Atm_apf1d:

```
Atm_apf3d <= -2      : isotropic scattering phase function
-2 < Atm_apf3d <= -1 : Rayleigh scattering phase function
-1 < Atm_apf3d < 1   : Henyey-Greenstein phase function with asymmetry parameter of Atm_apf1d
1 <= Atm_apf3d       : a tabulated phase function (given by the database file, see the namelist &link_anchor(mcarSca_nml_i
```

See Atm_apf1d for more details.

Data import procedure, common for binary and text formats

It is assumed that the file has multiple data sets:

```
data set 1
data set 2
...
```

Data set index to be read in for a computation job can be specified by Atm_idread in the namelist file.
The following Fortran code is used for data import for a single data set.

```
!####
  subroutine mcarAtm__read(mbswap) !* Import data from file *
    !
    !// Note this procedure touches only layers iz = [iz3l:iz3u].

    integer,  intent(in) :: mbswap
    integer :: ip, ikd, iz, iz3
    real(R_), allocatable :: omgp3d(:, :, :, :)

    ! Initialize
    allocate (omgp3d(Atm_nx, Atm_ny, Atm_nz3, Atm_np3d))

    ! Open & read in
    if (Atm_mfmt == 0) then ! text
       call mcarUtl__txtAtm_read(Atm_iud, Atm_nx, Atm_ny, Atm_nz3, Atm_nkd, Atm_np3d, &
            & Atm_mtprof, Atm_tmpa3d(:, :, Atm_iz3l:), Atm_abst3d(:, :, Atm_iz3l:, :), &
            & Atm_apfp3d(:, :, Atm_iz3l:, :), Atm_scap3d(:, :, Atm_iz3l:, :), omgp3d)
    else ! binary
       call mcarUtl__binAtm_read(Atm_iud, Atm_nx, Atm_ny, Atm_nz3, Atm_nkd, Atm_np3d, &
            & Atm_mtprof, Atm_tmpa3d(:, :, Atm_iz3l:), Atm_abst3d(:, :, Atm_iz3l:, :), &
            & Atm_apfp3d(:, :, Atm_iz3l:, :), Atm_scap3d(:, :, Atm_iz3l:, :), omgp3d, mbswap, Atm_idread)
    end if
    !// Atm_scap3d() represents tentatively the extinction coefficients, here.
    !// Atm_abst3d() represents tentatively the gaseous absorption coefficients, here.

    ! Scale gaseous absorption coefficients
    if (Atm_fabs3d /= 1.0_R_) Atm_abst3d(:, :, Atm_iz3l:Atm_iz3u, :) = &
         & Atm_abst3d(:, :, Atm_iz3l:Atm_iz3u, :) * Atm_fabs3d
```

```
    ! Bext & Omega --> Bsca & Babs
    do ip = 1, Atm_np3d
       do iz = Atm_iz3l, Atm_iz3u
          iz3 = iz - Atm_iz3l + 1
          do ikd = 1, Atm_nkd
             Atm_abst3d(:, :, iz, ikd) = Atm_abst3d(:, :, iz, ikd) + Atm_scap3d(:, :, iz, ip) &
                  & * (1.0_R_ - omgp3d(:, :, iz3, ip)) * Atm_fext3d(ip)
          end do
          Atm_scap3d(:, :, iz, ip) = Atm_scap3d(:, :, iz, ip) * omgp3d(:, :, iz3, ip) * Atm_fext3d(ip)
          !// Now, Atm_abst3d() represents total absorption coefficients.
          !// Now, Atm_scap3d() represents scattering coefficients.
       end do
    end do
    deallocate (omgp3d)

  end subroutine mcarAtm__read
```

As above, absorption and extinction coefficients are scaled by Atm_fabs3d and Atm_fext3d, respectively, just after data import. Subsequently extinction coefficients and single scattering albedos are converted to absorption and scattering coefficients.
One data set should have the following data streams (record lines), in a correct order:

data array	# of streams
Atm_tmpa3d(:,:,:)	Atm_nz3 or Atm_nz3+1
Atm_abst3d(:,:,:,:)	Atm_nz3 * Atm_nkd
Atm_extp3d(:,:,:,1)	Atm_nz3
Atm_omgp3d(:,:,:,1)	Atm_nz3
Atm_apfp3d(:,:,:,1)	Atm_nz3
Atm_extp3d(:,:,:,2)	Atm_nz3
Atm_omgp3d(:,:,:,2)	Atm_nz3
Atm_apfp3d(:,:,:,2)	Atm_nz3
...	Atm_nz3

For details, see below.

Binary format

Structure and format of the binary data file can be fully described by GrADS control file. An example is examples/les05_0045.atm.ctl:

```
DSET ^les05_0045.atm
*OPTIONS LITTLE_ENDIAN
*OPTIONS BIG_ENDIAN
OPTIONS template
TITLE les05_0045.atm
UNDEF 1.0e+37
XDEF        60 LINEAR 1 1
YDEF        60 LINEAR 1 1
ZDEF        22 LINEAR 1 1
TDEF         1 LINEAR 00:00Z00JAN0001 1YR
VARS         5
tmpa3d 22 99 Temperature perturbation (K)
abst3d 21 99 Absorption coefficient perturbation (/m)
extp3d 21 99 Extinction coefficient (/m)
omgp3d 21 99 Single scattering albedo
apfp3d 21 99 Phase function specification parameter
ENDVARS
```

When direct access binary format (Atm_mfmt=1), record length is set as nrec=nbyte*Atm_nx*Atm_ny, where nbyte is automatically set as 1 or 4, depending on Fortran compiler. The data are read in by the following Fortran codes.

```
!####
  subroutine mcarUtl__binAtm_read(iu, nx, ny, nz3, nkd, np3d, mtprof, &
       & tmpa3d, absg3d, apfp3d, extp3d, omgp3d, mbswap, idloc)
    !
    !* Read in 3-D atmosphere model parameters from a binary file *

    integer,  intent(in) :: iu
    integer,  intent(in) :: mtprof
    integer,  intent(in) :: nkd
    integer,  intent(in) :: np3d
    integer,  intent(in) :: nx, ny
    integer,  intent(in) :: nz3
    real(R_), intent(out) :: tmpa3d(:, :, :), absg3d(:, :, :, :)
    real(R_), intent(out) :: extp3d(:, :, :, :), omgp3d(:, :, :, :)
```

```
    real(R_), intent(out) :: apfp3d(:, :, :, :)
    integer, intent(in), optional :: mbswap ! flag for byte swapping (0=no, 1=yes)
    integer, intent(in), optional :: idloc  ! dataset location index
    real(R4_) :: wrk(nx, ny) !(AUTO)
    integer   :: ikd, ip3d, iz, iz3max, mswap, irec

    ! Initialize
    if (nz3 <= 0) return
    if (mtprof == 0) then
       iz3max = nz3
    else
       iz3max = nz3 + 1
    end if
    mswap = 0
    if (present(mbswap) .and. mbswap == 1) mswap = 4
    if (present(idloc)) then
       irec = (iz3max + nz3 * (nkd + 3*np3d)) * (idloc - 1) ! = (record index to be read) - 1
       call fileRec_set(iu, irec)
    end if

    ! Read in
    do iz = 1, iz3max ! temperature
       call bin_read_o2R4(iu, wrk, mswap=mswap)
       tmpa3d(1:nx, 1:ny, iz) = wrk(1:nx, 1:ny)
    end do
    do ikd = 1, nkd ! gaseous absorption coefficient
       do iz = 1, nz3
          call bin_read_o2R4(iu, wrk, mswap=mswap)
          absg3d(1:nx, 1:ny, iz, ikd) = wrk(1:nx, 1:ny)
       end do
    end do
    do ip3d = 1, np3d
       do iz = 1, nz3 ! extinction coefficients
          call bin_read_o2R4(iu, wrk, mswap=mswap)
          extp3d(1:nx, 1:ny, iz, ip3d) = wrk(1:nx, 1:ny)
       end do
       do iz = 1, nz3 ! single scattering albedo
          call bin_read_o2R4(iu, wrk, mswap=mswap)
          omgp3d(1:nx, 1:ny, iz, ip3d) = wrk(1:nx, 1:ny)
       end do
       do iz = 1, nz3 ! phase function specification parameter
          call bin_read_o2R4(iu, wrk, mswap=mswap)
          apfp3d(1:nx, 1:ny, iz, ip3d) = wrk(1:nx, 1:ny)
       end do
    end do

  end subroutine mcarUtl__binAtm_read
```

Text format

When sequential access text format (Atm_mfmt=0), the data block to be read is recognized after a line that contains the following line:

```
%mdla3d
```

After the signature is found, the data are read by the following Fortran codes:

```
!####
  subroutine mcarUtl__txtAtm_read(iu, nx, ny, nz3, nkd, np3d, mtprof, &
       & tmpa3d, absg3d, apfp3d, extp3d, omgp3d)
    !
    !* Read in 3-D atmosphere model parameters from a text file *

    integer,  intent(in) :: iu
    integer,  intent(in) :: mtprof
    integer,  intent(in) :: nkd
    integer,  intent(in) :: np3d
    integer,  intent(in) :: nx, ny
    integer,  intent(in) :: nz3
    real(R_), intent(out) :: tmpa3d(:, :, 1:), absg3d(:, :, :, :)
    real(R_), intent(out) :: extp3d(:, :, :, :), omgp3d(:, :, :, :)
    real(R_), intent(out) :: apfp3d(:, :, :, :)
    integer   :: ierr, ikd, ip3d, ix, iy, iz, iz3max

    ! Initialize
```

```
      if (nz3 <= 0) return
      if (mtprof == 0) then
         iz3max = nz3
      else
         iz3max = nz3 + 1
      end if
      iz = 0
      call skipToSign(iu, '%mdla3d', ierr)
      call err_read(ierr, iu, 'mcarUtl__txtAtm_read: %mdla3d is not found.')

      ! Temperature
      read (iu, *, err=1, end=9)
      do iz = 1, iz3max
         read (iu, *, err=1) ((tmpa3d(ix, iy, iz), ix = 1, nx), iy = 1, ny)
      end do

      ! Gaseous absorption coefficient
      do ikd = 1, nkd
         read (iu, *, err=2, end=9)
         do iz = 1, nz3
            read (iu, *, err=2) ((absg3d(ix, iy, iz, ikd), ix = 1, nx), iy = 1, ny)
         end do
      end do

      ! Cloud & aerosol parameters
      do ip3d = 1, np3d
         read (iu, *, err=3, end=9)
         do iz = 1, nz3 ! extinction coefficients
            read (iu, *, err=3) ((extp3d(ix, iy, iz, ip3d), ix = 1, nx), iy = 1, ny)
         end do
         read (iu, *, err=4, end=9)
         do iz = 1, nz3 ! single scattering albedo
            read (iu, *, err=4) ((omgp3d(ix, iy, iz, ip3d), ix = 1, nx), iy = 1, ny)
         end do
         read (iu, *, err=5, end=9)
         do iz = 1, nz3 ! phase function specification parameter
            read (iu, *, err=5) ((apfp3d(ix, iy, iz, ip3d), ix = 1, nx), iy = 1, ny)
         end do
      end do

      ! Exit
      return
1     call err_read(1, iu, 'mcarUtl__txtAtm_read: tmpa3d at iz =' //num2str_AN(iz))
2     call err_read(1, iu, 'mcarUtl__txtAtm_read: absg3d at iz =' //num2str_AN(iz))
3     call err_read(1, iu, 'mcarUtl__txtAtm_read: extp3d at iz =' //num2str_AN(iz))
4     call err_read(1, iu, 'mcarUtl__txtAtm_read: omgp3d at iz =' //num2str_AN(iz))
5     call err_read(1, iu, 'mcarUtl__txtAtm_read: apfp3d at iz =' //num2str_AN(iz))
9     call err_read(-1, iu, 'mcarUtl__txtAtm_read: Unexpected EOF. Array sizes would be invalid.')

    end subroutine mcarUtl__txtAtm_read
```

Edit

External input file Sfc_inpfile

This file should contain data of surface properties and can be in a text or binary format.
IMPORTANT NOTE : Binary format is highly recommended for faster data import and flexible random data accessibility. In the future versions, it is likely that only binary format files could be accepted.
Variables read in from this file are described below. Example data files can be found in the "examples" directory.
Numbers of elements should be specified in the namelist file:

```
Sfc_nxb, Sfc_nyb : # of surface elements along x and y axes
Sfc_npsfc : max # of surface parameters in a single surface element
```

Sfc_npsfc should be = [1,5], depending on Sfc_mbrdf, i.e. what BRDFs are used. Number of BRDF parameters differs by BRDF model kind. For example, if DSM model and Lambertian BRDFs are used (Sfc_mbrdf(1:4) = /(1,1,0,0)/, then Sfc_npsfc = 5, set automatically.

Sfc_tmps2d

```
real(R_), save, allocatable :: Sfc_tmps2d(:,:)    ! (nxb, nyb)
```

Surface temperature perturbation (K). Actual temperature will be

```
temp(:,:) = Sfc_tmp + Sfc_tmps2d(:,:)
```

Sfc_jsfc2d

```
integer,  save,  allocatable :: Sfc_jsfc2d(:, :)    ! (nxb, nyb)
```

2-D distribution of type index of surface BRDF, which is assumed as follows:

```
0: black
1: Lambertian
2: Flx_diffuse-specular mixture (DSM) model
3: Rahman-Pinty-Verstraete (RPV) model
4: Li-Sparse-Ross-Thick (LSRT) model
```

When the file is in binary format, this variable is read into a real(R4_) array (4-byte elements) tentatively, and then rounded to nearest neighbor integer values.

Sfc_psfc2d

```
real(R_),  save,  allocatable :: Sfc_psfc2d(:, :, :)  ! (nxb, nyb, Sfc_NPAR)
```

2-D distribution of surface BRDF parameters. Definitions are the same as Sfc_param.

Data import procedure, common for binary and text formats

```
!####
  subroutine mcarSfc__read(mbswap)  !* Import data from file *

    integer,  intent(in)  :: mbswap
    integer :: nmax,  i

    ! Initialize
    nmax = 0
    do i = 1,  4
       if (Sfc_mbrdf(i)  == 1)  nmax = max(nmax,  Sfc_npsfc(i))
    end do

    ! Read in
    Sfc_idloc = Sfc_idloc + 1
    if (Sfc_mfmt == 0)  then ! text
       call mcarUtl__txtSfc_read(Sfc_iud,  Sfc_nxb,  Sfc_nyb,  nmax,  Sfc_tmps2d,  Sfc_psfc2d,  Sfc_jsfc2d)
    else
       if (Sfc_idloc /= Sfc_idread)  Sfc_idloc = Sfc_idread
       call mcarUtl__binSfc_read(Sfc_iud,  Sfc_nxb,  Sfc_nyb,  nmax,  Sfc_tmps2d,  Sfc_psfc2d,  &
            & Sfc_jsfc2d,  mbswap,  Sfc_idloc)
    end if

  end subroutine mcarSfc__read
```

Binary format

Structure and format of the binary data file can be fully described by GrADS control file. An example is here:

```
DSET ^missing_example_file
*OPTIONS LITTLE_ENDIAN
*OPTIONS BIG_ENDIAN
OPTIONS template
TITLE An example
UNDEF 1.0e+37
XDEF       256 LINEAR 1 1
YDEF       256 LINEAR 1 1
ZDEF         5 LINEAR 1 1
* This example shows a case with Sfc_npsfc = 5,  which may depends on Sfc_mbrdf
TDEF         1 LINEAR 00:00Z00JAN0001 1YR
VARS         3
tmps2d  1 99 Temperature (K)
jsfc2d  1 99 BRDF type index
psfc2d  5 99 BRDF parameters (when Sfc_npsfc = 5)
ENDVARS
```

When direct access binary format (Sfc_mfmt=1), record length is set as nrec=nbyte*Sfc_nxb*Sfc_nyb, where nbyte is automatically set as 1 or 4, depending on Fortran compiler. The data are read in by the following Fortran codes.

```
!####
  subroutine mcarUtl__binSfc_read(iu, nxb, nyb, npsfc, tmps2d, psfc2d, jsfc2d, mbswap, idloc)
    !
    !* Read in surface model parameters from a binary file *

    integer,  intent(in) :: iu
    integer,  intent(in) :: nxb
    integer,  intent(in) :: nyb
    integer,  intent(in) :: npsfc
    integer,  intent(out) :: jsfc2d(:, :)
    real(R_), intent(out) :: tmps2d(:, :), psfc2d(:, :, :)
    integer, intent(in), optional :: mbswap ! flag for byte swapping (0=no, 1=yes)
    integer, intent(in), optional :: idloc  ! dataset location index
    real(R4_) :: wrk(nxb, nyb) !(AUTO)
    integer   :: ipsfc, mswap, irec

    ! Initialize
    mswap = 0
    if (present(mbswap) .and. mbswap == 1) mswap = 4
    if (present(idloc)) then
       irec = (2 + npsfc) * (idloc - 1) ! = (record index to be read) - 1
       call fileRec_set(iu, irec)
    end if

    ! Read in
    call bin_read_o2R4(iu, wrk, mswap=mswap)
    tmps2d(1:nxb, 1:nyb) = wrk(1:nxb, 1:nyb)
    call bin_read_o2R4(iu, wrk, mswap=mswap)
    jsfc2d(1:nxb, 1:nyb) = nint(wrk(1:nxb, 1:nyb))
    do ipsfc = 1, npsfc
       call bin_read_o2R4(iu, wrk, mswap=mswap)
       psfc2d(1:nxb, 1:nyb, ipsfc) = wrk(1:nxb, 1:nyb)
    end do

  end subroutine mcarUtl__binSfc_read
```

Text format

The data block to be read is recognized after a line that contains the following line:

```
%mdlsfc
```

After the signature is found, the data can be read in. The following Fortran code is used for reading data.

```
!####
  subroutine mcarUtl__txtSfc_read(iu, nxb, nyb, npsfc, tmps2d, psfc2d, jsfc2d)
    !
    !* Read in surface model parameters from a text file *

    integer,  intent(in) :: iu
    integer,  intent(in) :: nxb
    integer,  intent(in) :: nyb
    integer,  intent(in) :: npsfc
    integer,  intent(out) :: jsfc2d(:, :)
    real(R_), intent(out) :: tmps2d(:, :), psfc2d(:, :, :)
    integer   :: ierr, ipsfc, ixb, iyb

    ! Signature
    call skipToSign(iu, '%mdlsfc', ierr)
    call err_read(ierr, iu, 'mcarUtl__txtSfc_read: %mdlsfc is not found.')

    ! Surface model
    ipsfc = 0
    read (iu, *, err=1, end=2)
    read (iu, *, err=1, end=2) ((tmps2d(ixb, iyb), ixb = 1, nxb), iyb = 1, nyb)
    read (iu, *, err=1, end=2)
    read (iu, *, err=1, end=2) ((jsfc2d(ixb, iyb), ixb = 1, nxb), iyb = 1, nyb)
    read (iu, *, err=1, end=2)
    do ipsfc = 1, npsfc
       read (iu, *, err=1, end=2) ((psfc2d(ixb, iyb, ipsfc), ixb = 1, nxb), iyb = 1, nyb)
    end do

    return
1   call err_read( 1, iu, 'mcarUtl__txtSfc_read: ipsfc='//num2str_AN(ipsfc))
```

```
2    call err_read(-1, iu, 'mcarUtl__txtSfc_read: ipsfc='//num2str_AN(ipsfc))

  end subroutine mcarUtl__txtSfc_read
```

Edit

Next / Return

Next / Return

- 5. Radiative Transfer (RT) simulation
  - Examples
  - Usage of the "mcarats" executable
  - Computational speed
  - Possible problems
  - Tips using tools

Edit

5. Radiative Transfer (RT) simulation

Examples

The user can easily learn how to use the RT simulation tools from a shell script, ./examples/job_test, and the example I/O files in ./examples.

Usage of the "mcarats" executable

The binary executable for MCARaTS RT simulator is "mcarats".
The usage is as follows:

```
% [COMMAND_PATH/]mcarats ptot solver inpfile outfile
ptot   : Total number of incident photons.  Integer or real (e.g., 1000000 or 1.0e+6)
solver : Radiative transfer solver
  =0: F3D, Fully-3-Dimensional radiative transfer
  =1: P3D, Partially-3-Dimensional radiative transfer
  =2: ICA, Independent Column (or pixel) Approximation
inpfile : Namelist input file
outfile : Output file
```

For example, when the user operates the following: at "examples/" directory,

```
% ../bin/mcarats 1e5 0 conf_rs0067 test_rs0067
```

then the output data file, test_rs0067, will be generated. This output file is in a simple binary format. GrADS control file for the output will be automatically generated, with a filename "test_rs0067.ctl".

Computational speed

Here is an example for a case of radiance calculations:

```
Input namelist file: examples/conf_rs0067
Using a single CPU core
Radiance and LAMF calculation, visible wavelength, solar radiation
```

Test case	1	2
OS	Mac OS X	Linux (Fedora)
CPU	Intel Core 2 Duo 2.4 GHz	Intel Xeon 2.66 GHz
Compiler (options)	GNU Fortran (-O3)	Intel Fortran (-O -ipo)
Process speed (photons per second)	40,000	79,000

Speed is higher by about 20% if LAMF is not calculated. At absorbing wavelengths, the speed is higher.
Parallelization efficiency of the mcarats executable is very close to 100% for a large number of photons used, with 100 or more CPU cores. Therefore, the parallelizaed code has a good linearity in performance with number of CPU cores, for at least 100 or less CPU cores.

Possible problems

If you get some problem, please try to check the followings.
1) Invalid input format
The most frequent error is caused by invalid input format of the namelist file. If the code reported reading error of namelist input or external input files, then check the namelist file or the external input data files is written in valid format as described in Chapter 4.
For binary data to be read in, Wld_mbswap should be properly set.
2) Invalid input data values
The other common error is from invalid input data values. In this case, diagnostics might be reported from the codes (in some cases). Change the values for the input variables in valid ranges. Especially, input values for a variable should be integer if the variable is declared as integer. In addition, some variables should be in specific ranges.
3) Bug?
In other cases, there is no advise from developers. Please feel free to ask any question to developers, who hope to reply to all questions as long as there is enough time. If you find any bug, please report to the developers.

Tips using tools

To simulate for several similar cases with small differences, there is a useful tool,

```
./shl/replconfig.awk
```

that is an awk script. This tool can replace one parameter value in the namelist input file. A limitation is that the variable to be replaced should be scalar (i.e., not array variable). For example, we can replace the value for variable Rad_the in ./examples/conf_rs0067 by

```
% awk -f ./shl/replconfig.awk Rad_the 180 < ./examples/conf_rs0067 > ./conf
```

Then the new ./conf file contains a line as "Rad_the = 180". This is useful to perform many simulations, as follows:

```
% awk -f ./shl/replconfig.awk Rad_the 180 < ./examples/conf_rs0067 > ./conf
% ./bin/mcarats 1e5 0 ./conf ./out_the180
% awk -f ./shl/replconfig.awk Rad_the 160 < ./examples/conf_rs0067 > ./conf
% ./bin/mcarats 1e5 0 ./conf ./out_the160
% awk -f ./shl/replconfig.awk Rad_the 140 < ./examples/conf_rs0067 > ./conf
% ./bin/mcarats 1e5 0 ./conf ./out_the140
% rm -f ./conf
```

Edit

Next / Return

Next / Return

- 6. Output data from the "mcarats" codes
  - Quick visualization using GrADS
  - General format of the output file
  - Flux data output
  - Radiance data output

Edit

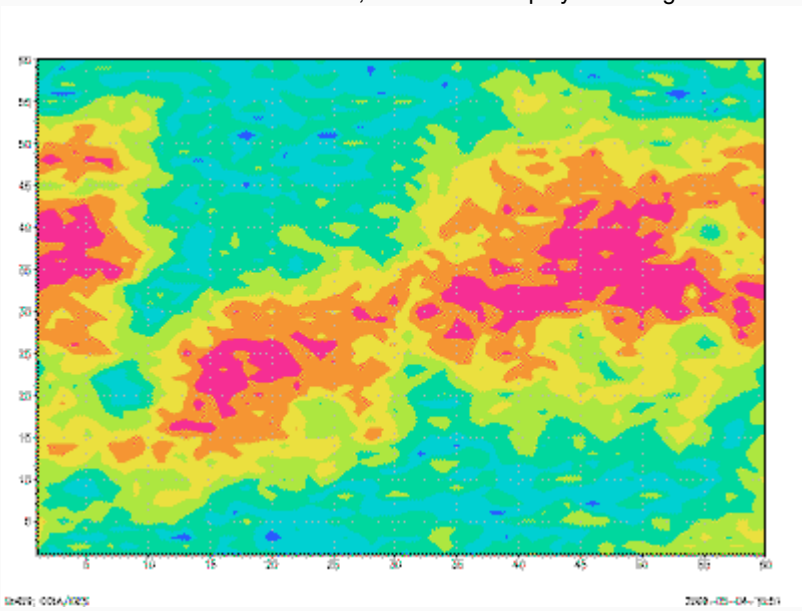
6. Output data from the "mcarats" codes

Quick visualization using GrADS

The output file from the mcarats executable is in a simple binary format. GrADS can be used to quickly visualize the results. If GrADS is installed in your system, you can try the followings:

```
% ./bin/mcarats 1e5 0 ./examples/conf_rs0067 out
% grads -l
...
ga-> open out.ctl
...
LON set to 1 60
LAT set to 1 60
LEV set to 1 1
Time values set: 1:1:0:0 1:1:0:0
E set to 1 1
ga-> set gxout shaded
ga-> set mproj off
ga-> set clevs 0.3 0.35 0.4 0.45 0.5 0.55 0.6
ga-> d a1
ga-> printim img_rs0067.png white
ga-> quit
```

When "d a1" command is entered, the window displays an image.



General format of the output file

The GrADS control files fully describe the output data. An example of GrADS control file, ./examples/out_rs0067.ctl:

```
DSET ^out_rs0067
*OPTIONS LITTLE_ENDIAN
*OPTIONS BIG_ENDIAN
OPTIONS template
TITLE out_rs0067
UNDEF 1.0e+37
XDEF        60 LINEAR 1 1
YDEF        60 LINEAR 1 1
ZDEF        41 LINEAR 1 1
TDEF         2 LINEAR 00:00Z00JAN0001 1YR
VARS         2
a1 1 99 Radiance
b1 41 99 Pathlength Statistics
```

```
ENDVARS
```

In the example above, the control file explains the followings:

- Data are stored in 4 dimensions
- Numbers of elements in X/Y/Z/T-dimensions are 60, 60, 41, and 2, respectively
- Two variables are stored: One is radiance, and another is pathlength statistics
- The radiance array is in 3-dimension: (60, 60, 2)
- The pathlength array is in 4-dimension: (60, 60, 41, 2)

In this example, dimension X/Y/Z corresponds to spatial X/Y/Z dimension. Dimension T corresponds to different sources (with solar directions). Pathlength statistics are here layer air mass factor, which is average pathlength in a layer, normalized by each layer geometrical thickness. Output quantity and number of elements depends on how the user sets up the namelist parameters.
Pseudo Fortran code to read in the data above:

```
real(4) :: radi(60, 60, 2)
real(4) :: lamf(60, 60, 41, 2)
open (10, file='out_rs0067', access='direct', recl=1*60*60)
!// Note recl depends on compiler. Should be 4*60*60 sometimes.
do isrc = 1, 2
  read (10) radi(:, :, isrc)
  do iz = 1, 41
    read (10) lamf(:, :, iz, isrc)
  end do
end do
```

Flux data output

An example of flux and heating rate calculation result is examples/out_hr0213. GrADS control file:

```
DSET ^out_hr0213
*OPTIONS LITTLE_ENDIAN
*OPTIONS BIG_ENDIAN
OPTIONS template
TITLE out_hr0213
UNDEF 1.0e+37
XDEF        60 LINEAR 1 1
YDEF        60 LINEAR 1 1
ZDEF        41 LINEAR 1 1
TDEF         2 LINEAR 00:00Z00JAN0001 1YR
VARS         4
a1 2 99 Flux Density
a2 2 99 Flux Density
a3 2 99 Flux Density
b1 41 99 Heating Rate
ENDVARS
```

The three flux variables, a1, a2, and a3, in the above respectively denote solar direct beam flux, total downward flux, and upward flux.
In this example, fluxes at top and bottom of atmosphere were calculated, and heating rates were calculated in full 3-D atmospheric cells, by setting the namelist parameters:

```
Wld_mtarget = 1
Flx_mflx = 1
Flx_mhrt = 1
```

These control what quantities are calculated and output. If Flx_mflx = 3, for example, then fluxes at all layer boundaries will be output.
Pseudo Fortran code to read in the data above:

```
real(4) :: flx(60, 60, 3, 2)
real(4) :: hrt(60, 60, 41, 2)
open (10, file='out_hr0213', access='direct', recl=1*60*60)
!// Note recl depends on compiler. Should be 4*60*60 sometimes.
do isrc = 1, 2
  do i = 1, 3
    read (10) flx(:, :, i, isrc)
  end do
  do iz = 1, 41
    read (10) hrt(:, :, iz, isrc)
  end do
end do
```

Radiance data output

Radiance calculation mode (Wld_mtarget = 2) and visualization mode (Wld_mtarget = 3) produce output file in a similar format. An example of GrADS control file:

```
DSET ^out_ci0067
*OPTIONS LITTLE_ENDIAN
*OPTIONS BIG_ENDIAN
OPTIONS template
TITLE out_ci0067
UNDEF 1.0e+37
XDEF       640 LINEAR 1 1
YDEF       480 LINEAR 1 1
ZDEF         1 LINEAR 1 1
TDEF         6 LINEAR 00:00Z00JAN0001 1YR
VARS         1
a1 1 99 Radiance
ENDVARS
```

Pseudo Fortran code to read in the data above:

```
real(4) :: rad(640, 480, 6)
open (10, file='out_ci0067', access='direct', recl=1*640*480)
!// Note recl depends on compiler. Should be 4*640*480 sometimes.
do it = 1, 6
  read (10) rad(:, :, it)
end do
```

Edit

Next / Return

Next / Return

- 7. Utilities
  - Introduction
  - bin_couple
  - bin_convert
  - bin_exposure
  - bin_gray
  - bin_dump
  - bin_sample
  - bin_stat
  - row_stat

Edit

7. Utilities

Introduction

The output file of mcarats code is in a simple binary format, which is easily treated with GrADS. Several utility tools are provided to process the data files. It is not difficult to make a new tool to process the data files. If you made a useful tool, you can contribute to the project by providing it.

- bin_couple to do a simple calculation with a couple of binary data files
- bin_convert to convert a binary data with a simple equation
- bin_dump to convert binary data to a text format
- bin_exposure to diagnose recommended exposure data for image generation
- bin_gray to generate gray scale images from a binary data file
- bin_sample to (re)sample data from a binary data file
- bin_stat to calculate spatial statistics of binary-format data
- row_stat to calculate spatial statistics of text-format data

When operating GrADS data files, control files are assumed to exist at the same directory as the data files.

bin_couple

This is used to do a simple calculation with a couple of binary data files.

```
usage: bin_couple method thresh inCtlFile1 inCtlFile2 outfile
 method:
    0 (sum)              : in1 + in2
    1 (difference)       : in1 - in2
    2 (product)          : in1 * in2
    3 (ratio)            : in1 / in2
    4 (relative diff. 1) : (in1 - in2) / in2
    5 (relative diff. 2) : 2*(in1 - in2) / (in1 + in2)
    6 (square diff.)     : in1*in1 - in2*in2
    7 (root square diff.): sqrt(in1*in1 - in2*in2)
    8 (diff. square)     : (in1 - in2)*(in1 - in2)
    9 (root diff.)       : sqrt(in1) - sqrt(in2)
 thresh: threshold of the denominator when method=3-5.
         If the denominator is less than the threshold, then the output value is reset
         at a very small value (-1.0E+30).
 inCtlFile1 : GrADS control file for input data file 1
 inCtlFile2 : GrADS control file for input data file 2
 outfile : Output data file
```

The output file is in the same format as the input. Numbers of data elements and dimension sizes should be the same between the two input files.

bin_convert

This is used to convert binary data with a simple equation.

```
usage: bin_convert nx ny method A B inCtlFile outfile
 nx       : # of x-dimension elements for output.
 ny       : # of y-dimension elements for output.
 method=0 : v = u               (does nothing)
        1 : v = A + B * u       (linear)
        2 : v = A * u^B         (power-law)
        3 : v = exp(A + B*u)    (exponential)
        4 : v = ln(A + B*u)     (logarithmic)
```

```
        5 : v = A + (1 + B*G)*u (random noises are superimposed)
            where G is a normalized Gaussian noise (average: 0; variance:1).
 inCtlFile : GrADS control file for an input file
```

If nx*ny=0, then these are reset as defined in the input file. Otherwise the averaging is operated.
The output file is in the same format (binary) as the input, but if nx or ny is smaller than the input, then the amount of output data is reduced.

bin_exposure

This computes recommended exposure data, which can be used as Rmax values for bin_gray utility. For multiple data files (e.g., each RGB channel), average exposure can be calculated. If data for multiple time steps are available, time series of exposure values can be convolved by a filter with a user-specified time width and time lag. The time lag is useful to realistically simulate that real optical instrument and human eyes adjust their aperture size with a time delay.
This tool would be useful as a utility for image sequence generation, for example.

```
usage: bin_exposure timeLag timeWid fmax ctlFile1 (ctlFile2 ctlFile3 ...)
 timeLag : time lag (integer)
 timeWid : temporal filter width (integer, 1 means no filtering)
 fmax    : factor for automatic determination of Rmax (usually around 1.2)
 ctlFile : GrADS control files for binary data
```

See also bin_gray for how obtained data can be used.

bin_gray

This generates gray scale images in a simple 1-byte-per-pixel format.

```
usage: bin_gray factor Rmax power controlFile outfileHead (iz ivar it) (< exposureFile)
 controlFile: GrADS control file for the input data file
 outfileHead: filename head for output files
 Scaling formula:    Y = min(Rmax, X * fac)
 Conversion formula: Z = int(Y / Rmax**pwr * 255) = [0, 255]
 If Rmax == 0, then Rmax is read in from standard input (exposureFile).
 If iz, ivar, and it are given, only the specified data will be processed.
```

Appropriate exposure data can be pre-determined and recorded in the exposureFile. If Rmax=0, then the pre-determined exposure data (Rmax values) will be read in from the stdin and used. A tool, bin_exposure, was made for this purpose.
Multiple output files will be generated if multiple data sets are stored in the input file. One output file should contains one image data.
Example of usage: at examples/ directory,

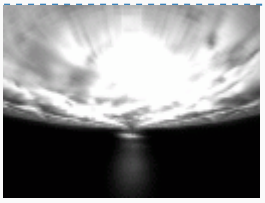
```
$ ../bin/bin_gray 1.182 1.1 0.75 out_ci0045.ctl img_ci0045
        400         300   1.1000000     img_ci0045_tl_var1_zl.gray
        400         300   1.1000000     img_ci0045_tl_var2_zl.gray
```

Generated files will be listed with numbers of pixels and Rmax values. If ImageMagick is installed and "convert" command is available, then you can try

```
$ convert -depth 8 -size 640x480+0 img_ci0045_tl_var1_zl.gray img_ci0045_tl_var1_zl.png
```

This will generate a PNG format image file. Then you can view the image, by

```
$ display -rotate 180 img_ci0045_tl_var1_zl.gif
```



bin_dump

This converts binary data to a text format.

```
usage : bin_dump inCtlFile > outfile
 inCtlFile : GrADS control file for an input file
 outfile   : output file (text format)
```

Results are printed in the standard output. Example of usage:

```
$ bin/bin_dump examples/out_rs0067.ctl | less
# it, ivar, n :           1           1           1
 0.43441936       0.39797837       0.37406304       0.43220648       0.42556950  ...
```

```
 ...
```

bin_sample

This is used to resample data from a binary data file. This is useful to extract partial data from the file.

```
usage: bin_sample ixi ixs nx iyi iys ny inCtlFile outfile
 ixi      : initial index for X
 ixs      : step of X-index
 nx       : # of x-pixels for output
 iyi      : initial index for Y
 iys      : step of Y-index
 ny       : # of y-pixels for output
```

An example is here: at examples/ directory,

```
% ../bin/bin_sample 2 2 30 2 2 30 out_rs0067.ctl sample_rs0067
```

In this case, original data for 60x60 pixels are available, and output result from the above will be for 30x30 pixels.

bin_stat

This is used to calculate basic statistics of binary-format datasets.

```
usage : bin_stat inCtlFile (mmask thresh1 thresh2) > outfile
 inCtlFile : input GrADS control file
 outfile   : output file (text format)
 mmask=0 : no data masking
       1 : dat > thresh1
       2 : dat < thresh1
       3 : thresh1 < dat < thresh2
       4 : dat < thresh1 or dat > thresh2
```

Output data file reports fraction of data possibly masked by the threshold(s), minimum, maximum, average, standard deviation, root-mean-square (rms), and skewness of read data set. One set of data corresponds to X*Y two dimensional array, which is defined in the control file. The output are printed to stdout.
Usage example:

```
$ bin/bin_stat examples/test_rs0067.ctl
# iz,  ivar,    it,   frac,    min,         max,         average,     stdev,       rms,         skewness
    1     1     1   1.0000  3.3940E-01  6.8978E-01  4.8468E-01  7.1038E-02  4.8986E-01   4.2904E-01
    1     2     1   1.0000  3.6220E-01  2.2394E+00  1.0513E+00  4.2648E-01  1.1345E+00   8.8130E-01
   ...
```

row_stat

This is used to calculate basic statistics of a text data sequence.

```
usage: row_stat N func ithresh (thresh1 thresh2) < infile > outfile
 N         : # of sequential data for calculating statistics,  in a line
 func      : 0,  raw data; 1,  ln(dat); 2,  exp(dat);
             The conversion is performed before the statistics computation.
 ithresh
   =0 : no threshold,
   =1 : dat >= thresh1,
   =2 : dat <= thresh1,
   =3 : thresh1 <= dat <= thresh2,
   =4 : dat <= thresh1 or dat >= thresh2
```

Output data file reports fraction of data filtered by the threshold(s), minimum, maximum, average, standard deviation, root-mean-square (rms), and skewness of read data set. One set of data is read in from a line in the input.

Usage example:

```
$ bin/bin_dump examples/out_rs0067.ctl > tmp
$ bin/row_stat 3600 0 0 < tmp
#    frac,    min,       max,      average,    stdev,      rms,      skewness
   1 1.0000  3.0509E-01  7.0611E-01  4.8130E-01  7.7988E-02  4.8757E-01  2.8613E-01
   2 1.0000  3.7049E-01  2.2713E+00  1.0711E+00  4.4416E-01  1.1595E+00  9.5030E-01
   ...
```

The output are printed to stdout. Statistics for data sequences, each one of which contains 3600 data, are reported in this example.

Edit

Next / Return

- 8. Using MCARaTS
  - Examples in the package
    - Radiation budget applications
    - Remote sensing applications
    - Visualization applications
  - Photorealistic visualization of 3-D cloud scene
    - Total sky imager simulation
    - Sunset scenes
    - Aerosol effects
    - Radiance mapping
  - Pathlength analyses
    - Column air mass factors (CAMFs)
    - Radiance histogram wrt total pathlength

Edit

8. Using MCARaTS

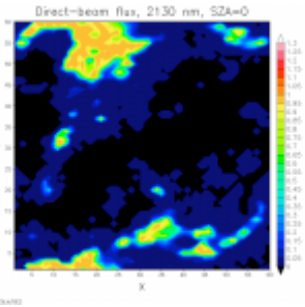
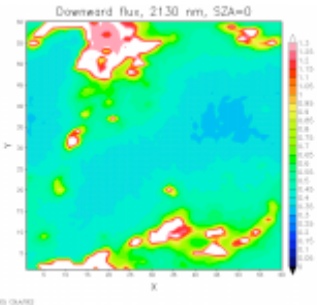
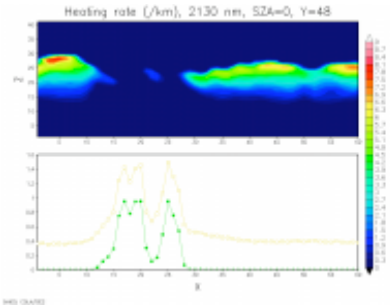
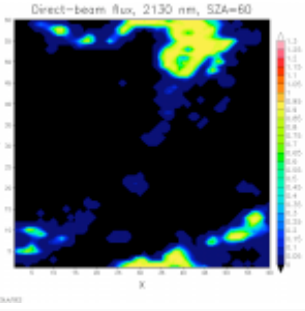
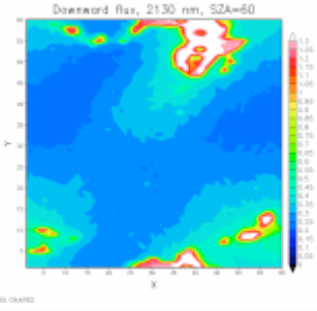
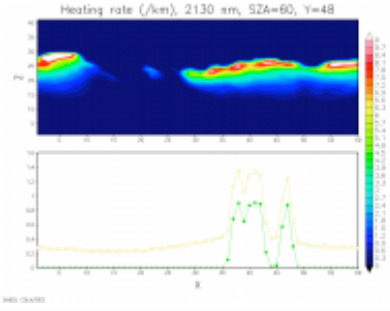
Examples in the package

Radiation budget applications

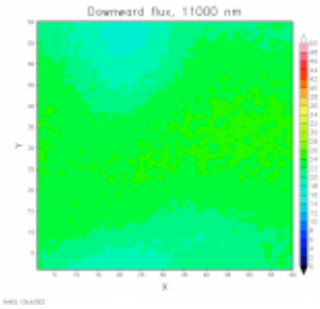
Input files: conf_hr0213 and conf_hr1100.
These are flux and heating rate calculation examples for two wavelengths, 2130 nm and 11000 nm. Spatial distribution of cloud water density for stratocumulus cloud scene was taken from a large eddy simulation. Rayleigh scattering and typical aerosol vertical profile are assumed. Surface reflection is represented by Lambertian model. Simulations are performed for two solar zenith angles (SZAs). Fluxes at top and bottom of the atmosphere and 3-D heating rates are computed.
For these cases, important variables in the namelists are listed as follows:

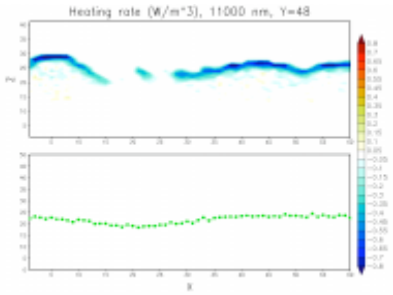
```
 Wld_mtarget = 1 ! target mode (1=flux&HR, 2=radiance, 3=volume rendering)
 Src_nsrc = 2    ! two SZAs
 Flx_mflx = 1    ! fluxes at TOA and BOA
 Flx_mhrt = 1    ! 3D heating rates
 Sfc_mtype = 1   ! Lambertian
 Src_the = 180.0, 120.0 ! zenith angles of the source transport directions
```

Results are shown below.

Case	Direct-beam flux	Downward flux	Heating rate and fluxes
2130 nm, SZA = 0			
2130 nm, SZA = 60			

11000 nm -



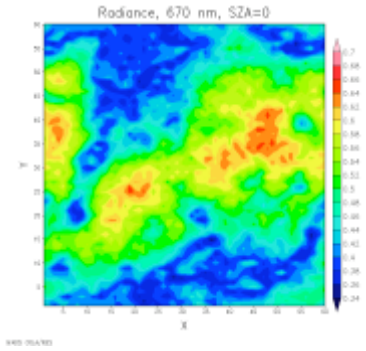
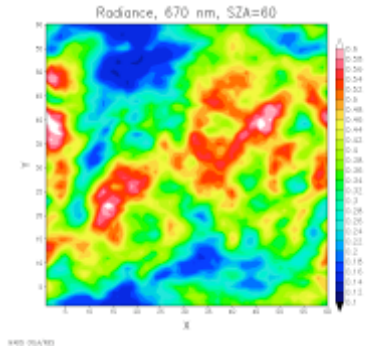
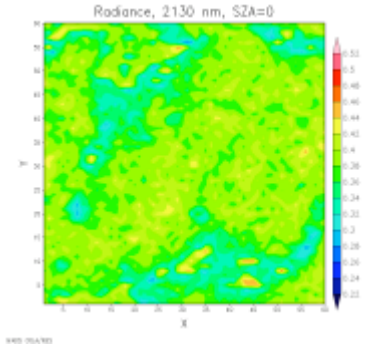
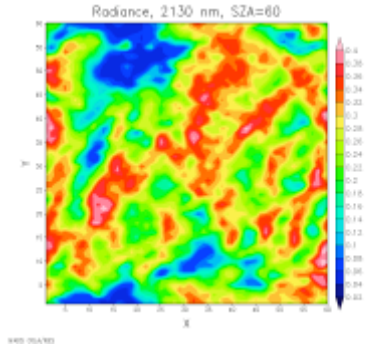
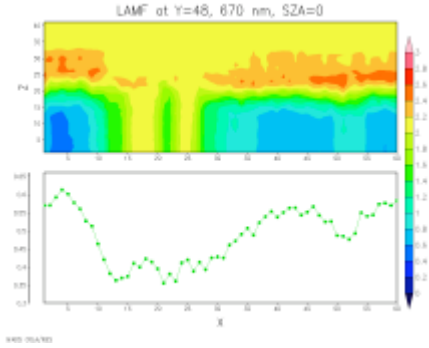
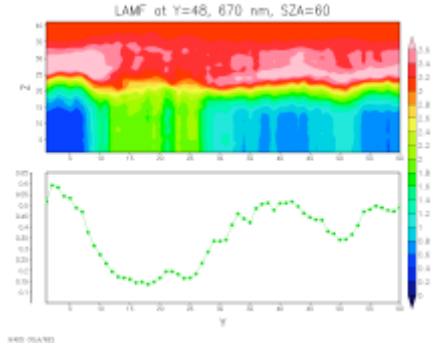


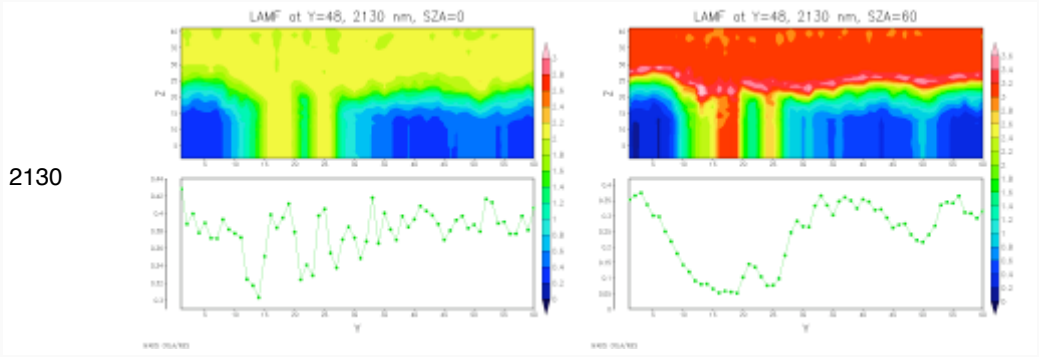
Remote sensing applications

Input files: conf_rs0067 and conf_rs0213.
These are radiance calculation examples for two wavelengths, 670 nm and 2130 nm. The atmosphere is synthesized as the radiation budget examples. Surface reflection is represented by RPV BRDF model. Simulations are performed for two solar zenith angles (SZAs) at the near-infrared wavelength. Normalized nadir radiances and layer air mass factors are computed.
For these cases, important variables in the namelists are listed as follows:

```
Wld_mtarget = 2 ! target mode (1=flux&HR, 2=radiance, 3=volume rendering)
Src_nsrc = 2    ! two SZAs
Rad_mrkind = 2  ! 2nd kind of radiance
Rad_mplen = 1   ! LAMFs are calculated
Rad_nrad = 1    ! one radiance direction
Sfc_mtype = 3   ! RPV BRDF
Src_the = 180.0, 120.0 ! zenith angles of the source transport directions
Rad_the = 180.0   ! nadir looking
Rad_zloc = 1.0e+5 ! from space
```

Results are shown below.

Wavelength	SZA = 0	SZA = 60
670		
2130		
670		



2130

Visualization applications

Input files: conf_ci0045,conf_ci0055, conf_ci0067
These are visualization examples for theree wavelengths, 450 nm, 550 nm, and 670 nm. The atmosphere is synthesized as the radiation budget examples. Surface reflection is represented by DSM BRDF model. Simulations are performed for one solar zenith angle (SZA). Normalized quasi-radiances are computed by the rendering method 4, the highest-quality method. Two images are generated. One view point is near the surface, and another is at slightly above the cloud bottom altitude. Wide field-of view camera is assumed.
For these cases, important variables in the namelists are listed as follows:

```
Wld_mtarget = 3 ! target mode (1=flux&HR, 2=radiance, 3=volume rendering)
Src_nsrc = 1    ! one SZA
Rad_mrkind = 1  ! the lst kind of radiance
Rad_mpmap = 1   ! rectangular pixel mapping
Rad_nrad  = 2   ! two radiance images
Rad_nxr   = 400 ! # of pixels - U
Rad_nyr   = 300 ! # of pixels - V
Vis_mrend  = 4  ! RTE-based rendering with TMS correction
Sfc_mtype = 2   ! DSM BRDF
Src_the   = 120.0 ! zenith angle of incident beam transport direction
Rad_xpos  = 5*0.45
Rad_ypos  = 5*0.5
Rad_zloc  = 95, 305 ! altitudes of view points
Rad_the   = 75, 90  ! zenith angles of the FOV center directions
Rad_phi   = 0, 0    ! azimuth angles of the FOV center directions
Rad_psi   = 5*90    ! image rotation around the FOV center
Rad_umax  = 5*140   ! FOV U size
Rad_vmax  = 5*110   ! FOV V size
Rad_qmax  = 5*180.0
```

Generated data can be converted to gray scale images, using a utility code bin_gray. Then, using some ImageMagick tools, true color images can be generated. An example shell script, ./examples/job_colorimg.csh, is available as a solution:

```
#!/bin/csh -f

## To generate color images from binary (GrADS compatible)  data files

# Arguments
if ($#argv < 6) then
    echo "$0 blue.ctl green.ctl red.ctl outfileHead pwr rmax (timeLag timeWid fmax)"
    exit 2
endif
set imgc=./$argv[4]
set pwr=$argv[5]
set rmax=$argv[6]

# User variables
set fcol=(1.28 1.0 0.8) # color balance

# System variables
set bin_exposure=../bin/bin_exposure
set bin_gray=../bin/bin_gray
set tmp0=./$$.tmp.0
set imgs=(./.tmp_img_B ./.tmp_img_G ./$$.tmp_img_R)
set imgt=./$$.tmp_img.tif

# Gray images
if ($rmax == 0) then # automatic exposure
    echo $bin_exposure $argv[7] $argv[8] $argv[9] $argv[5] $argv[1] $argv[2] $argv[3]
    $bin_exposure $argv[7] $argv[8] $argv[9] $argv[5] $argv[1] $argv[2] $argv[3] > $tmp0
    foreach c(1 2 3)
    $bin_gray $fcol[$c] $rmax $pwr $argv[$c] $imgs[$c] < $tmp0
```

```
    end
else # manual exposure
    foreach c(1 2 3)
    $bin_gray $fcol[$c] $rmax $pwr $argv[$c] $imgs[$c]
    end
endif
set nx=(`grep XDEF $argv[1] | awk '{print $2}'`)
set ny=(`grep YDEF $argv[1] | awk '{print $2}'`)
set nz=(`grep ZDEF $argv[1] | awk '{print $2}'`)
set nt=(`grep TDEF $argv[1] | awk '{print $2}'`)
set nv=(`grep VARS $argv[1] | awk '{print $2}'`)

# Color images
@ t = 1
while($t <= $nt)
    if ($t <= 9) then
    set tstr="000"$t
    else if ($t <= 99) then
    set tstr="00"$t
    else if ($t <= 999) then
    set tstr="0"$t
    else
    set tstr=$t
    endif
    @ v = 1
    while($v <= $nv)
    @ z = 1
    while($z <= $nz)
        if ( -r $imgs[1]"_t"$t"_var"$v"_z"$z".gray" ) then
       set str="_t"$t"_var"$v"_z"$z
       set strl="_t"$tstr"_var"$v"_z"$z
       set pwrl=$pwr
       @ ntry = 0
       while($ntry < 30) # loop for trials
           ### This is because the gray image file is sometimes broken.
           set err=0
           foreach c(1 2 3)
          convert -depth 8 -size $nx"x"$ny+0 $imgs[$c]$str".gray" $imgs[$c]$str".png"
          if ($status != 0) then
              set err=1
              echo Error: $imgs[$c]$str".gray" "with pwrl="$pwrl
          endif
           end
           if ($err == 0) then # success
          set ntry=1000000
           else # error, then retry
          set pwrl=`echo $pwrl"*1.00333" | bc`
          foreach c(1 2 3)
              if ($rmax == 0) then # automatic exposure
             $bin_gray $fcol[$c] $rmax $pwrl $argv[$c] $imgs[$c] $z $v $t < $tmp0
              else # manual exposure
             $bin_gray $fcol[$c] $rmax $pwrl $argv[$c] $imgs[$c] $z $v $t
              endif
          end
           endif
           @ ntry = $ntry + 1
       end
       composite -compose CopyGreen $imgs[2]$str".png" $imgs[3]$str".png" $imgt
       composite -compose CopyBlue  $imgs[1]$str".png" $imgt $imgc$strl".tif"
       echo $imgc$strl".tif" : complete
       rm -f $imgs[1]$str* $imgs[2]$str* $imgs[3]$str* $imgt
        endif
        @ z = $z + 1
    end
    @ v = $v + 1
    end
    @ t = $t + 1
end
rm -f $tmp0
```

An example of usage:

```
./job_colorimg.csh out_ci0045.ctl out_ci0055.ctl out_ci0067.ctl img_ci 0.6 0 0 1 2
```

Color images in TIFF format will be generated. They can be viewed by, for example, display command (an ImageMagick tool). Result color images are shown below.

z=95
near surface

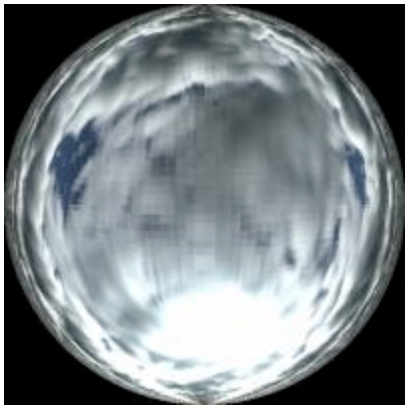
z=305
near cloud bottom

Photorealistic visualization of 3-D cloud scene

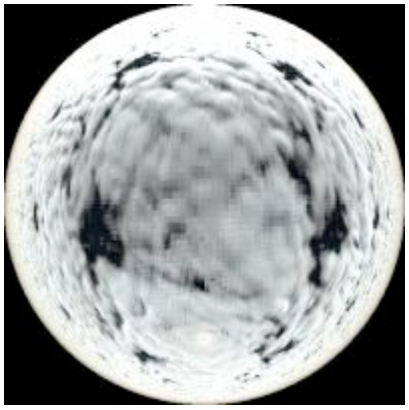
As shown above, MCARaTS can be used as a visualization tool. We can virtually observe 3-D cloud field from various view points. By modifying the atmospheric and surface properties in the example cases, various images can be generated as follows.

Total sky imager simulation

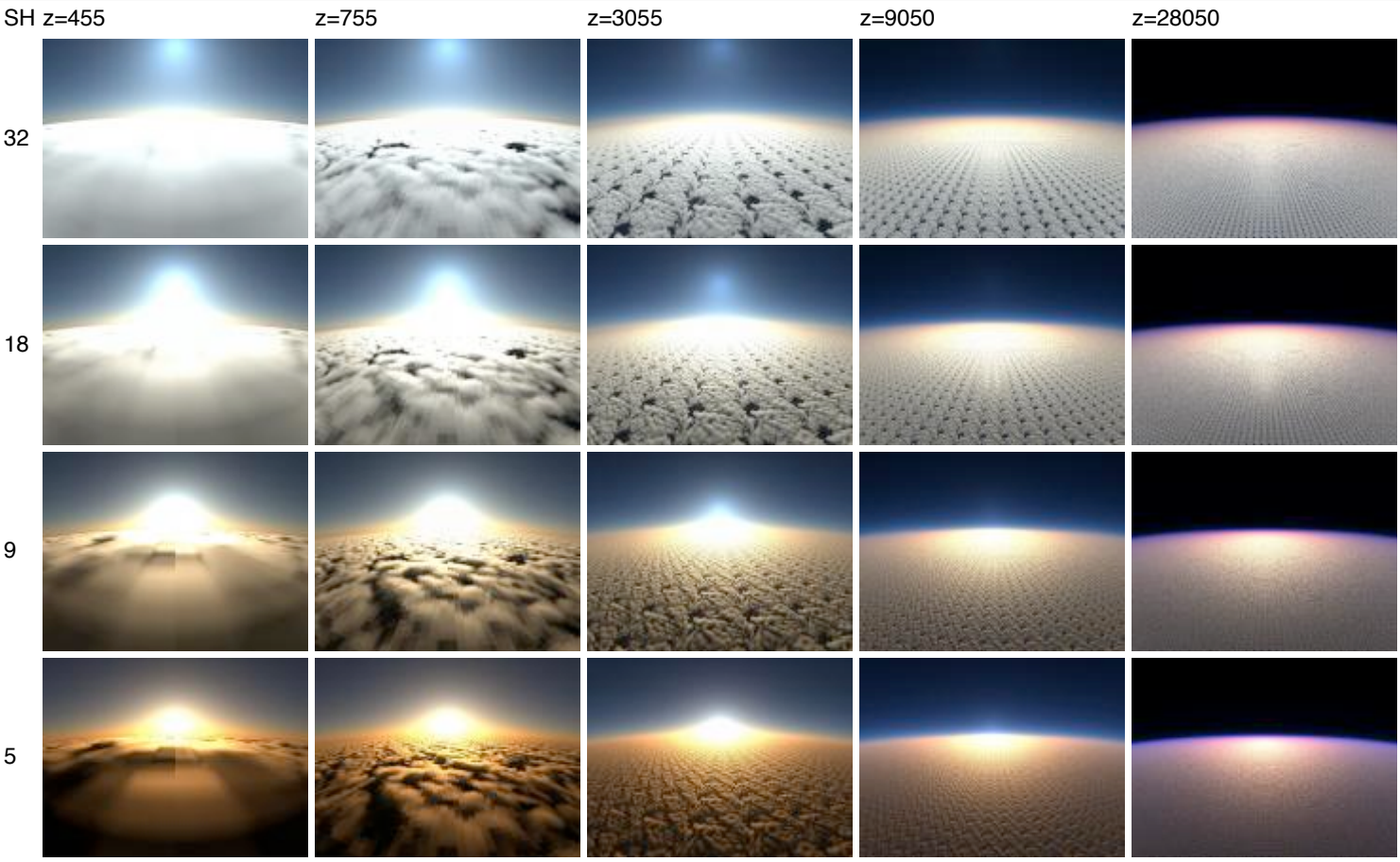
Upward, near surface



Downward, above cloud



Sunset scenes



3

(SH: Solar height (deg.))

Aerosol effects

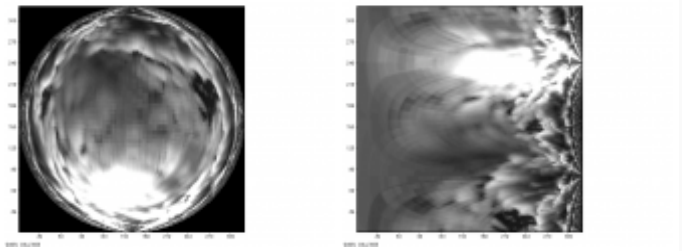
Images were obtained by increasing and decreasing aerosol density by a factor of 3.

Aerosol*(1/3)	Aerosol*1	Aerosol*3

Radiance mapping

For Rad_mrkind = 1 or 3, one of two angular-pixel mapping methods can be chosen. A target mode Wld_mtarget can be 2 or 3. Image axes U and V are differently defined for the two methods.
The following two examples both show hemispherical distribution of radiance looking a cloud deck from ground.

Rad_mpmap=1	2
Rectangular	Polar

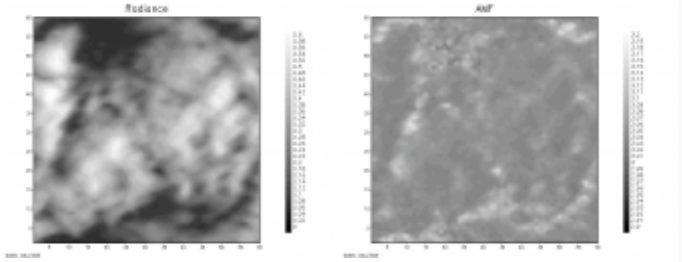


Pathlength analyses

Column air mass factors (CAMFs)

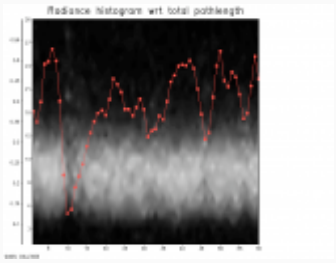
Can be calculated for each radiance pixel. Weighting functions for CAMFs can be specified by the user. In the following example, the weighting function was uniform within the entire atmosphere. We can see a negative correlation between radiance and CAMF, implying that the pathlength becomes longer at cloud gaps.
In more practical applications, the weighting function should correspond to some gas profile (e.g., O2 or O4) of interest.

Radiance	CAMF



Radiance histogram wrt total pathlength

Can be calculated for each radiance pixel. The total pathlength is calculated as a sum of all photon flight paths, from radiation source to the detector. The total pathlength bins are specified by the user, with minimum and maximum values and the number of bins. This provides a time-resolved histogram, so that lidar signals can be simulated with localized radiative source, for example. The following figure is plotted for a section at y=30, with solar radiative source.



Edit

Next / Return

Return

- 9. References

Edit

9. References

Iwabuchi, H., 2006: Efficient Monte Carlo methods for radiative transfer modeling. Journal of the Atmospheric Sciences, 63, No. 9, 2324– 2339.

Iwabuchi, H., and H. Kobayashi, 2008: Modeling of radiative transfer in cloudy atmospheres and plant canopies using Monte Carlo methods. FRCGC Technical Report No. 8, 199 pp, 2008.

Iwabuchi, H., T. Suzuki: Fast and accurate radiance calculations for cloudy atmospheres using truncation approximation. In Press, Journal of Quantitative Spectroscopy & Radiative Transfer, doi:10.1016/j.jqsrt.2009.04.006, 2009.

Lucht W., C. B. Schaaf, A. H. Strahler, 2000: An algorithm for the retrieval of albedo from space using semiempirical BRDF models. IEEE Trans. Geosci. Remote Sens. 38, 977– 998.

Marchuk GI, Mikhailov GA, Nazaraliev MA, Darbinjan RA, Kargin BA, Elepov BS. The Monte Carlo methods in atmospheric optics. Berlin: Springer Series in Optical Sciences, Springer-Verlag, 208 pp, 1980.

Nakajima, T., and M. Tanaka, 1983: Effect of wind-generated waves on the transfer of solar radiation in the atmosphere-ocean system, J. Quant. Spectrosc. Radiat. Transfer, 29, 521-537.

Nakajima, T., and M. Tanaka 1988: Algorithms for radiative intensity calculations in moderately thick atmospheres using a truncation approximation. J. Quant. Spectrosc. Radiat. Transfer, 40, 51– 69.

Rahman, H., B. Pinty, and M. M. Verstraete, 1993: Coupled surface-atmosphere reflectance (CSAR) model. 2. Semiempirical surface model usable with NOAA advanced very high resolution radiometer data, J. Geophys. Res., 98, 20,791– 20,801.

Edit

Return